VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Katedra informačních technologií

Autonomní řídicí systém FRC automobilu
Autonomous Control System for FRC Car

2010 Radek Tesař

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Radek Tesař**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Autonomní řídicí systém FRC automobilu
Autonomous Control System for FRC Car

Zásady pro vypracování:

Práce se bude zabývat vývojem řídicího systému malého soutěžního automobilu FRC. Úkolem v soutěži je zajet co nejrychleji stanovený počet kol na neznámé trati. Profil trati je možné určit pomocí snímačů. Cílem práce je sestavit stavebnici řídicího systému automobilu, navrhnout a realizovat ovládací algoritmus pro řídicí mikroprocesor.

V práci se zaměřte:
1. Sestavení stavebnice řídicího systému.
2. Možnosti snímání pohybu a polohy.
3. Výběr vhodných doplňujících snímačů.
4. Detekce profilu trati a polohy na trati.
5. Návrh algoritmu řízení pohybu.
6. Programování mikroprocesoru.
7. Zhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

1. BREJL, Milan. *Freescale Race Challenge 2010* [online]. 2010 [cit. 2010-10-26]. Dostupný z WWW: <http://hw.cz/FRC2010>.
2. BORENSTEIN, L., EVERETT, H. R., FENG, L. *Where am I? : Sensors and Methods for Mobile Robot Positioning.* J. Borenstein. [s.l.] : [s.n.], 1996. 282 s.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jiří Kotzian, Ph.D.**

Datum zadání: 20.11.2009

Datum odevzdání: 07.05.2010

doc. Dr.Ing. Eduard Sojka
vedoucí katedry

prof. Ing. Ivo Vondrák, CSc.
děkan fakulty

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Ostrava, 05. května 2010

.....

Podpis

Poděkování

Rád bych na tomto místě poděkoval všem, kteří mi z prací pomohli ať cennými radami, návrhy nebo nápady, protože bez nich by tato práce nevznikla. Jmenovitě bych rád poděkoval vedoucímu mé práce ing. Jiří Kotzianovi, Ph. D. za doporučení a pomoc při vývoji i psaní mé práce, ing. Milanu Brejlovi, Ph. D. za neutuchající entuziasmus při organizaci soutěže FRC, mgr. Janě Trojkové, Ph. D., za konzultace a pomoc s fyzikální podstatou práce, oponentovi mé práce ing. Michalu Krumniklovi za rady a nápady týkající se zpracování obrazu, všem soutěžícím, se kterými jsme vedli nekonečné diskuze nad problematikou řízení auta, rodičům, kteří se mnou měli trpělivost a všem ostatním, kteří se mi sem nevešli.

ABSTRAKT:

Tato práce se zabývá vývojem řídicího systému malého soutěžního automobilu FRC. Úkolem v soutěži je zajet co nejrychleji stanovený počet kol na neznámé trati. Profil trati je možné určit pomocí různých snímačů. Cílem práce je sestavit desku plošných spojů řídicího systému automobilu, navrhnout a realizovat ovládací algoritmus pro řídicí mikroprocesor. Automobil řídí 32 bitový mikroprocesor Coldfire V1 firmy Freescale Semiconductor, vývojové prostředí použité pro vývoj je Code Warrior Development Studio for Microcontrollers verze 6.3 speciální edice. K ukládání dat se využívá standardní micro SD kartu, formátovanou systémem FAT16. V počátcích jsem využil předpřipravený příklad a některé hotové komponenty tak, abych se mohl zaměřit na vlastní vývoj aplikace, měření údajů poskytovaných jednotlivými senzory a vyhodnocení naměřených hodnot.

KLÍČOVÁ SLOVA:

Gyroskop, akcelerometr, mikroprocesor, algoritmus vyhledávání textu, mapování dráhy, měření zrychlení, řízení DC motoru, PWM modulace, působení fyzikálních sil v zatáčce na hmotný bod.

ABSTRACT:

This work deals with development control system for a small car FRC. The challenge in the competition is run as quickly as possible a set number of rounds in an unknown route. Line profile of the route can be determined using various sensors. The goal is to populate car control system printed circuit board, design and implement control algorithms for microprocessor control. Vehicle use a 32-bit microprocessor, Coldfire V1 made by Freescale Semiconductor. Integrated Development Environment for development is Code Warrior for Microcontrollers Development Studio, Special Edition, version 6.3. For store measured data is used a standard SD card, formatted FAT16. In the beginning I used prepared example, and some finished components so I could focus on developing my own applications, measurement data provided by various sensors and the evaluation of the measured values.

KEYWORDS:

Gyroscope, accelerometer, microprocessor, text search algorithm, route mapping, measuring accelerate, DC motor control, PWM modulation, action the physics forces in the bend on a mass point.

BGA - ball grid array- pro zmenšení rozměrů součástek a zvýšení vstupně výstupních pinů se používá bezvývodová technologie. Spočívá na pouřití rastru plošek na spodní části pouzdra součástky v rozteči od 0,3 do 1 mm. Tyto plošky jsou pak propojeny s deskou plošných spojů cínovými kuličkami přesně definovaného průměru.

Bootloader – zavaděč. Program, který slouží k nahrání a spuštění aplikace v pracovní paměti mikropočítače. V případě počítačů IBM – PC kompatibilních tuto úlohu přebírá bios.

Corilisova síla - je setrvačná síla působící na tělesa, která se pohybují v rotující neinerciální vztažné soustavě tak, že se mění jejich vzdálenost od osy otáčení. Coriolisova síla má směr kolmý ke spojnici těleso - osa otáčení a způsobuje stáčení trajektorie tělesa proti směru otáčení soustavy

Debugování - Odstranění chyb vzniklých při tvorbě programového vybavení

EMWA – Exponential Weighted Moving Average. Průměrování s exponenciálním zapomínáním.

Fine Pitch - malá rozteč vývodů více vývodové součástky, typicky méně než 0,5 mm.

IDE – integrované vývojové prostředí pro vývoj cílové aplikace, většinou slouží pro psaní kódu, překlad, nahrání aplikace do cílového mikropočítače a případně ladění (debugování) takto vytvořeného kódu.

Krokování kódu - technika debugování kódu, používá se nejčastěji u jednočipových mikropočítačů. Mikropočítač je připojen k hostitelskému PC se spuštěným IDE prostřednictvím ladícího (debugovacího) interface. Většina výrobců MCU používá JTAG nebo vlastní interface. Coldfire V1 firmy Frescale Semiconductors používá BDM interface.

MCU, Mikrokontrolér, mikroprocesor, jednočipový mikropočítač – v mé práci je takto nazván řídící procesor elektroniky auta. Odborníci doufám prominou určité zobecnění pojmů, které však v kontextu této práce není na závadu.

Normálové zrychlení - zrychlení při pohybu po trajektorii, které působí ve směru normály (ve směru od středu kružnice po které se bod pohybuje).

Přerušení (mikropočítače) - asynchronně vykonávaná rutina, kterou je nutno provést v co nejkratším časovém intervalu. Převedeno do běžného života- čtu si knihu, v tom zazvoní telefon (přerušení). Do knihy si dám záložku (uložím údaj do zásobníku), vyřídím hovor (obsloužím přerušení), otevřu knihu na místě které jsem si založil (ze zásobníku vyjmu uložený údaj) a pokračuji ve čtení (hlavním programu).

PWM - pulse wave modulation, pulsně šířková modulace.

RTC - Real time counter- čítač přesného času- slouží k vytváření například hodin, stopek atd.

Tečné zrychlení - zrychlení při pohybu po trajektorii, které působí ve směru tečny (ve směru pohybu bodu, pokud by na něj nepůsobila dostředivá síla).

OBSAH

ÚVOD

Práce se bude zabývat vývojem řídicího systému malého soutěžního automobilu FRC. Úkolem v soutěži je zajet co nejrychleji stanovený počet kol na neznámé trati. Profil trati je možné určit pomocí snímačů. Cílem práce je sestavit stavebnici řídicího systému automobilu, navrhnout a realizovat ovládací algoritmus pro řídicí mikroprocesor. V práci jsem se zaměřil nejprve na sestavení vzorové desky plošných spojů, osazení a oživení elektroniky tak, jak byla dodána firmou Freescale Semiconductors.

Hardwarovou platformu, její osazení a problémy s oživením nastiňuji v kapitole 1. Dále zde popisuji úpravy a implementaci nových senzorů, problémy spojené s jejich implementací, omezení dané soutěžními podmínkami. Také zde uvádím princip soutěže a stručné pravidla, aby si čtenář mohl udělat celkovou představu.

V kapitole 2 - Teorie působení sil při jízdě rozebírám základní fyzikální principy a síly působící na auto při jízdě rovnoměrnou ustálenou rychlostí. Dále zde navrhuji použití vhodných senzorů pro měření takových sil a princip činnosti uvedených senzorů.

V nejrozsáhlejší třetí kapitole nazvané vývoj software řeším základní problémy spojené s vývojem software pro auto - od jednotlivých algoritmů pro měření vzorků dat, jejich filtrování, řízení motoru auta, způsob hledání profilu neznámé trati a následně jízdu podle nalezeného profilu. Významnou částí je využití přerušovacího subsystému, ve kterém se odehrává měření a vzorkovaní dat všech senzorů, dále pak regulace PWM pro řízení motoru auta.

V poslední, čtvrté kapitole s názvem vyhodnocení testů shrnuji výsledky všech pokusů prováděných v průběhu celé práce. Hodnotím zvlášť hardwarovou platformu, software a dále navrhuji řešení nedostatků a zlepšení pro další kola soutěže v následujících letech.

.

1 HARDWAROVÁ PLATFORMA

Freescale Semiconductor ČR a Freescale Rumunsko, s podporou distributora autodráhy Carrera, firmou ConQuest entertainment, s.r.o, pořádali pro studenty elektrotechnických vysokých škol u nás, na Slovensku a v Rumunsku již druhý ročník soutěže Freescale Race Challenge (FRC), která má za cíl vdechnout mezi vývojáře mikroprocesorových aplikací závodní atmosféru. Úkolem zúčastněných studentů bude postavit a naprogramovat auto na autodráhu, které zajede nejkratší čas na 10+10 okruhů na neznámé trati.

Toto auto však nebude nikdo řídit - přesněji řečeno bude je řídit mikroprocesor. Během prvního okruhu musí jet opatrně a trať si "osahat". Až rozpozná, že už jede druhý okruh, může podle předchozího mapování trati již očekávat, kdy přijde jaká zatáčka a kdy naopak rovinka.

V napájecích kolejnicích je stabilní napětí, mikroprocesor auta si sám řídí rychlost. Mapování trati je založeno na použití akcelerometru, gyroskopu nebo jiných senzorů, které měří odstředivou sílu působící na auto v zatáčkách.

Tato soutěž mě zaujala, hlavě proto, že se jedná o komplexní řešení problému – je nutno vytvořit jak funkční hardware tak pro něj napsat vhodný software. Nejedná se však o nějaký uměle vytvořený problém, ale o skutečnou funkční aplikaci, jejíž životaschopnost se projeví v závodě. Tam bude možno jednoznačně porovnat jednotlivé nápady, myšlenky a řešení, i jejich zpracování. Já jsem se však rozhodl řešit v rámci své bakalářské práce navíc porovnání údajů z různých senzorů a jejich vyhodnocení. Hlediskem pak není jen přesnost naměřených dat, ale i jejich opakovatelnost v rámci průjezdů jednotlivými koly tratě a využitelnost při on-line zpracování a schopnost mikroprocesoru s těmito daty pracovat.

Pravidla soutěže jsou dvojí a to pro hlavní závod a pro vyřazovací závod (viz. [5]). Pravidla hlavního závodu popisují, že závodí každý zvlášť na čas. V prvním kole se jezdí 10 okruhů v pravé dráze, ve druhém kole 10 okruhů v levé dráze. Součet časů obou kol určuje výsledné pořadí. Autíčko lze položit na dráhu kdekoliv a čas se začne počítat až při prvním průjezdu časomírou. Při vypadnutí autíčka jej smí jeho majitel (v případě týmu jen jeden z týmu) nasadit. Základní vlastností závodní dráhy je, že dráha je až do začátku závodu všem soutěžícím neznámá. Ví se pouze tolik, že bude poskládána z dílů univerzitní dráhy, na které jsme trénovali. Nebudou na ní žádná svodidla ani krajnice, jen zatáčky stanovených poloměrů (dle setu dodaného každé univerzitě), rovinky a také překřížení drah. Smyslem předpisů pro závodní auta je zaměřit studenty více na vývoj inteligence auta a jeho algoritmu, než na mechanická vylepšení. Pravidla předepisují použít karoserii, podvozek, motor, gumy a vodící díl dodaného auta Audi R8 Carrera Evolution (obrázek 1). Přítlačné magnety je nutné z auta odmontovat.

Bezdrátové ovládání je zakázáno. Na autě může být pouze jeden přepínač umožňující výběr jednoho ze dvou módů. Jeho využití je libovolné. Auto může být vybaveno jakoukoliv další elektronikou nad rámec referenční platformy, avšak vše se musí vejít dovnitř karoserie. Dále je celková hmotnost auta omezena maximem 125 gramů.

Obrázek 1-1 Model auta Carrer Evolution Audi R8

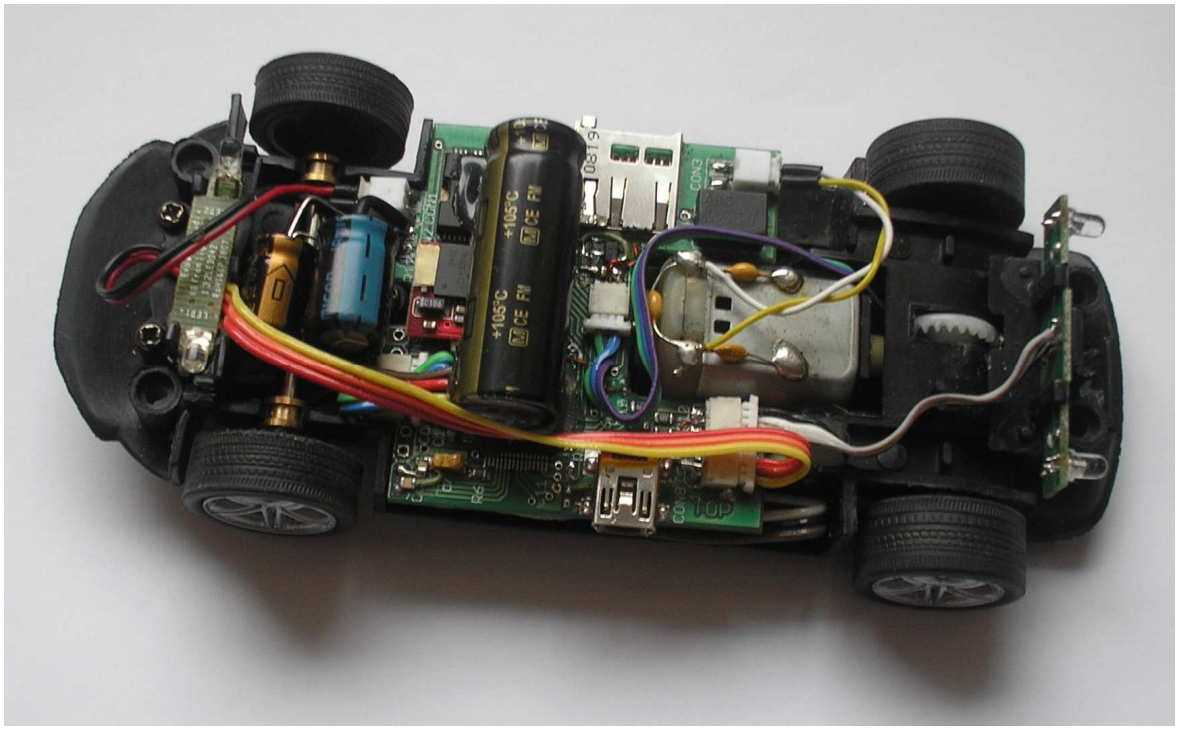
Pravidla vyřazovacího turnaje popisují, že vždy závodí 2 auta spolu, rychlejší postupuje, až po celkového vítěze. Soutěžící autíčka umístí na přesně určená místa na dráze a závod se startuje zapnutím napájení do dráhy. Vypadlé auto lze nasadit zpět stejně jako v hlavním závodě. Pokud se auta srazí na křížení drah, to, které vedlo (nejméně o půl kola), vyhrává. Vlastnosti závodní dráhy jsou stejné jako pro hlavní závod, navíc, tvar dráhy musí být symetrický, tak aby oba závodníci projížděli tvarem i délkou přesně stejnou dráhu. Předpisy pro auta jsou stejná jako pro hlavní závod, takže se stejným autem se lze zúčastnit obou závodů. Rozdíl je pouze v tom, že ve vyřazovacím závodě jsou dva napájecí díly, které jsou ovšem také umístěny symetricky. Toto je důležité pro některé způsoby detekce počátku nebo konce kola.

Pro hlavní závod se tedy předpokládá, že auto v prvním (neměřeném) kole bude mapovat tvar dráhy a následně podle této mapy pojede. V loňském kole existovaly tři různé způsoby řízení auta:

- ✓ řízení bezpečnou rychlostí
- ✓ řízení vyšší rychlostí se zrychlováním na rovinkách a brzděním v zatáčkách podle okamžitého normálového zrychlení. Někteří tento systém otočili, takže přidávali do zatáčky a na rovinkách jezdili nižší rychlostí
- ✓ v prvním kole mapování trati, při dalších kolech jízda maximální rychlostí na rovinkách, s brzděním před zatáčkou (tak jako v reálném automobilu)

První dvě kategorie reagují pouze na okamžité hodnoty naměřené senzory (většinou akcelerometrem), třetí využívá znalosti trati získané mapováním v prvním kole tak, aby autíčko projelo trať maximální možnou rychlostí. Výsledek prvního ročníku soutěže jasně ukázal, že třetí kategorie je jednoznačně nejrychlejší, ovšem vyladění autíčka je velmi problematický úkol

Každý účastník soutěže obdržel základní balíček podpory, který obsahoval základní součástky, desku plošných spojů pro demoverzi aplikace a model auta Audi R8. K těmto součástkám si musí každý účastník zajistit další nutné komponenty a případně rozšiřující moduly nebo senzory podle vlastního uvážení. Dále byla vytvořena vzorová aplikace pro oživení elektroniky a první pokusy.



Obrázek 1-2 Podvozek auta s instalovanou již hotovou elektronikou

1.1 Původní elektronika Freescale Semiconductors

V balíčku podpory se tedy nachází standardní auto na autodráhu Carrera Evolution Audi R8 (obrázek 1), tištěný spoj - referenční hardwarová platforma, součástky k jeho osazení (kromě rezistorů, kondenzátorů a diod), 32bitový mikroprocesor MCF51JM64, 3D akcelerometr MMA7361, integrovaný H-můstek MC33931, mini USB konektor, micro SD patice, 8.000MHz krystal, 1A diodový můstek DB106S a 3.3V lineární napěťový regulátor.

Pro nahrávání aplikace do autíčka slouží mini USB konektor na autíčku. V dodaném mikroprocesoru je již předem naprogramovaný USB-bootloader. Po připojení k PC se autíčko tváří jako vyměnitelná síťová jednotka, na kterou lze nakopírovat soubor, generovaný IDE Code Warrior při překladu aplikace.

Po odpojení od USB a postavení na dráhu se tato aplikace spustí. Toto řešení umožňuje jednoduché a rychlé nahrání aplikace, neumožňuje však debugování a krokování kódu. To může být jistým omezením při začátcích vývoje, v další fázi vývoje již auto na dráze krokovat nelze.

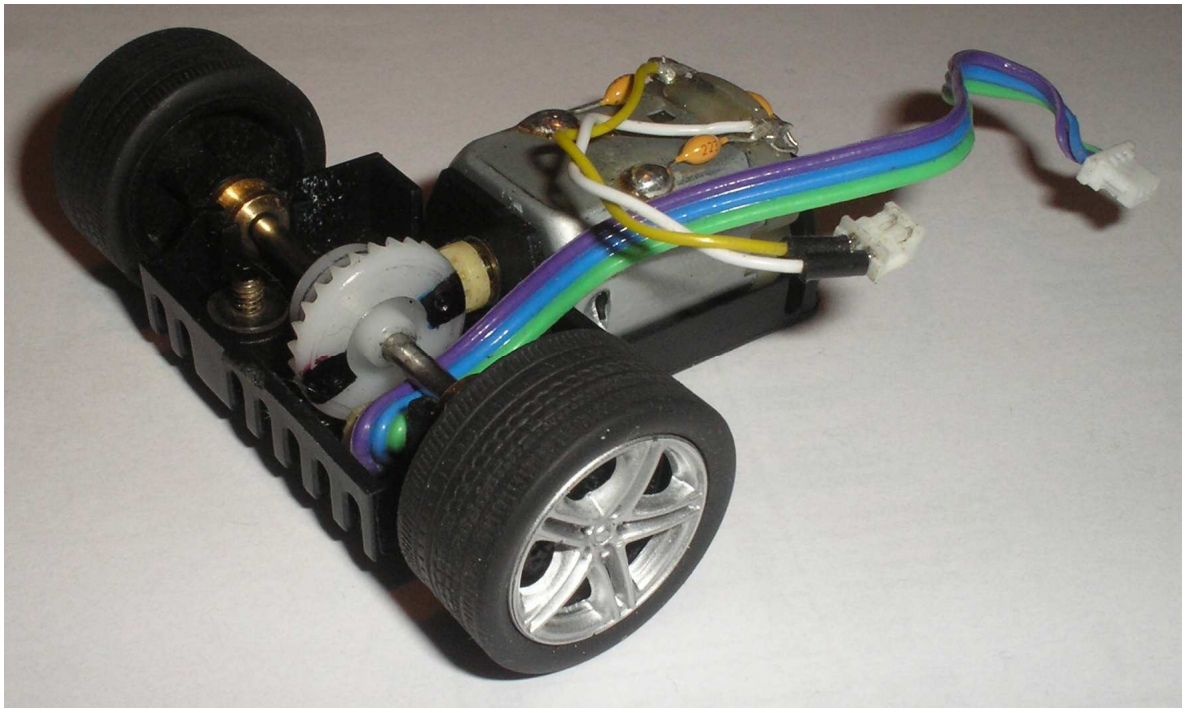
1.1.1 Zkušenosti s osazením a oživením původní elektroniky

Řada účastníků soutěže měla v počátcích problémy s osazením a oživením auta (viz. Diskuze na [5]). Osazení referenční platformy v mém případě proběhlo bez problémů. Použité součástky (např. H-můstek nebo mikroprocesor) jsou sice ve třídě fine pitch, což vyžaduje jisté zkušenosti s osazováním takových obvodů, ale to nebyl žádný problém, protože disponuji technologiemi pro osazování jak fine pitch tak BGA obvodů. Jako větší problém se ukázalo zprovoznění USB- bootloaderu. Po řadě více či méně neúspěšných pokusů jsem se rozhodl použít standardní BDM interface pro nahrání a ladění aplikace. Tento krok byl velmi šťastný, což se projevilo hlavně při ladění jednotlivých podprogramů a testování některých algoritmů. Při této příležitosti jsem vyměnil mikroprocesor za nejvyšší v řadě, MCF51JM128, který se liší hlavně zvětšenou pamětí flash.

1.2 Úpravy původní hardwarové platformy

Dodaný set je velmi dobře vybaven pro daný účel a je na něm možno postavit funkční software bez výrazných zásahů. Toto však nebylo účelem mé práce. Já jsem se zaměřil hlavně na porovnání různých senzorů a jejich využitelnost v rámci dané aplikace. Celkové provedení kompletní elektroniky zastavěné do podvozku auta je patrné z Obrázek 1-2.

1.2.1 Plánované úpravy auta a elektroniky

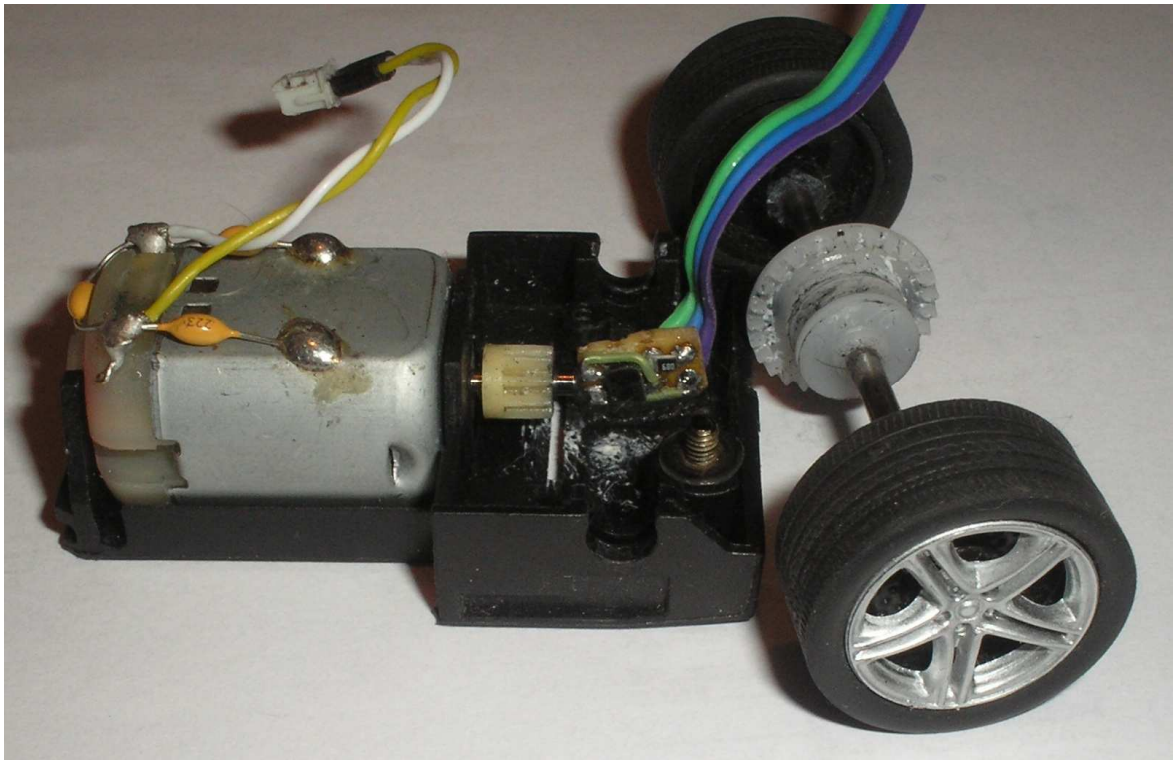


Obrázek 1-3 Vymontovaný motor s detailem přerušovacích značek optického systému

Pro mé účely jsem musel rozšířit původní návrh o následující senzory:

- ✓ optoelektronický snímač otáček
- ✓ optoelektronický snímač průjezdu cílovou rovinkou
- ✓ gyroskop
- ✓ bezkontaktní magnetický enkodér

Použité optoelektronické snímače jsou firmy Kingbright, typ KTIR0711S. Pracují v infračerveném spektru a zvolil jsem je hlavně kvůli jejich miniaturním rozměrům. Praktické provedené snímače otáček je na obrázcích Obrázek 1-3 a Obrázek 1-4. Největším problémem však bylo, že jsem se v průběhu celého vývoje potýkal s vyladěním senzorů. Jeden snímač je použit pro měření otáček zadních kol, druhý pak pro test průjezdu cílovou rovinkou. Snímač otáček zadních kol snímá přerušení paprsku čidla při otáčení hnacího ozubeného kola na zadní nápravě. K tomuto přerušení dojde 2x za jednu otočku, proto měříme dvojnásobek frekvence otáčení kola. Snímač průjezdu cílovou rovinkou je instalován na podvozku auta a snímá povrch vozovky (dráhy). Tento povrch je na celé dráze černý, jen v jediném místě (na cílové rovince) má svislou bílou čáru. Tato rovinka je součástí napájecího dílu, proto musí být na dráze vždy přítomna a je vždy jen jedna (mimo vyřazovacích turnajů kde jsou napájecí díly dva).



Obrázek 1-4 Detail provedení snímače otáček kol. Zde jsou patrné miniaturní rozměry snímače

Použitý gyroskop je jednoosý analogový gyroskopický senzor měřící otáčení v ose Z (yaw) z produkce firmy ST Microelectronics. Výstupem gyroskopického senzoru je přímo úhel otočení ve stupních za sekundu, čímž dostaneme relevantní informaci o otáčení i v ustálené zatáčce (např. na klopené dráze, nebo v letadle, kde například akcelerometr neukáže žádnou výchylku).

Bezkontaktní magnetický enkodér AS5045 od firmy Austriamicrosystem AG vyžaduje pro svou funkci jednoduchý dvoupólový magnet, rotující nad středem čipu. Poskytuje absolutní hodnotu otočení ve 4096 pozicích na otáčku, což je rozlišení 0,0879 stupně. Tato hodnota je poskytována ve formě digitálního sériového řetězce nebo PWM signálu. Moje myšlenka byla umístit tento senzor nad osu otáčení slotu auta a tak měřit úhel natočení slotu vůči ose auta.

Na trati se také může nacházet křížení, kdy auto přejede z jedné dráhy do druhé. V tomto místě je proto z technologických důvodů přerušeno vedení napájení. Důsledkem tohoto přerušení je pak restart procesoru a hlavně ztráta informací uložených v paměti RAM. Při malých rychlostech dokonce docházelo k úplnému zastavení auta, kdy jeho setrvačnost nestačila překonat místo, kde sběrače ztratily kontakt s napájecí lištou. Dalšími úpravami proto bylo například zvětšení kapacity filtračního kondenzátoru, nebo přidání kondenzátoru 3,3G/16V v provedení LOW ESR na vstup H-můstku. Všechny tyto úpravy měly za úkol zlepšit průjezd křížením dráhy, kde docházelo ke ztrátě napájení a k vybití kondenzátorů v průběhu tohoto přerušení. Mechanické úpravy auta jsem omezil na minimum, většinou se jednalo pouze o vyvedení konektorů, vytvoření otvoru pro čtečku micro SD karty, integrace

optického senzoru do podvozku a podobně. Po diskuzi s ostatními týmy závodu v Rožnově p. Radhoštěm jsem zjistil, že některé týmy zkoušely další úpravy, například odpružení přední nápravy auta a podobně. Problém však byl v tom, že se jednalo o nepovolené úpravy a nelze je pro závod použít.

1.2.2 Omezení úprav pro splnění podmínek soutěže

Investoval jsem značné pracovní i finanční úsilí do vytvoření vhodného držáku magnetu a modulu bezkontaktního magnetického enkodéru AS5045, přesto se mi nepodařilo je do auta implementovat tak aby splňovalo podmínky závodu. V podmínkách závodu je totiž klauzule, že nelze provádět takové úpravy, které by přesahovaly původní obrys auta, jinými slovy, z auta nesmí nic vyčuhovat. Ani při nejlepší vůli a použití CNC obráběcích strojů se nepodařilo vytvořit držák snímače a magnetu tak malý aby se vešel pod kapotu auta. Díky tomu jsem byl nucen tento velmi nadějný senzor vyřadit, což mě velmi mrzelo. Magnetický enkodér byl mým favoritem- poskytuje okamžitou informaci o úhlu natočení bez nutnosti cokoliv počítat nebo filtrovat, signál z něj není ničím rušen, ani zašuměn, proto jsem jej považoval za ideální pro tyto účely. Takže pro závod byly použity pouze optoelektronické snímače, gyroskop a akcelerometr.

Dalším omezením bylo nemožnost provádět jakékoliv úpravy podvozku a také omezení hmotnosti auta na 125 gramů. Základním pravidlem byla také nutnost demontovat přítlačné magnety na podvozku auta, čímž se výrazně sníží třecí síla působící na auto. V konečném důsledku to pak znamená nutnost dokonale vyladit řídící algoritmus auta.

2 TEORIE PŮSOBENÍ SIL PŘI JÍZDĚ

Předem této kapitoly bych rád uvedl, že podklady pro většinu fyzikální teorie jsem čerpal z konzultací s Mgr. Janou Trojkovou, vyučující fyziku v institutu fyziky a naší emailové korespondence, dále pak z ([3],[4]). Bohužel po dokončení práce jsem neměl dostatek času k tomu, abych nechal moji konzultantku nahlédnout do této práce k její recenzi. Proto veškeré zobecnění a nepřesnosti v této kapitole ať už úmyslné nebo neúmyslné, jsou chyby autora. Prosím proto o shovívavost odbornou veřejnost z oblasti fyziky, neboť fyzika je pro mne v této práci prostředkem k dosažení cíle, nikoliv cílem samotným.

2.1 Fyzikální princip chování auta při jízdě na dráze

Na auto jedoucí po dráze se můžeme dívat jako na bod pohybující se v kartézské soustavě souřadnic v dvourozměrném prostoru. V tomto prostoru je pak každý bod projeté dráhy určen souřadnicemi x a y. Pro zjednodušení budeme uvažovat, že se auto může pohybovat pouze v prvním kvadrantu, a to tam kde souřadnice x a y nabývají kladných hodnot. Z výše uvedeného je pak jasné, že množina bodů projeté dráhy vytváří trať (projetý okruh) a auto se bude vždy pohybovat po tomto okruhu, přestože tvar okruhu může být libovolný. Budeme-li první bod při rozjezdu auta považovat za bod B_0 se souřadnicemi (x, y), pak se po projetí jednoho (a každého dalšího kola) ocitneme přesně v tomtéž bodě. Naším cílem tedy je, najít efektivní algoritmus, který nám bude mapovat projetou dráhu a pohyb v tomto dvourozměrném prostoru. Tím pak získáme představu o tvaru dráhy v průběhu pohybu po této dráze a budeme schopni řídit rychlost průjezdu dráhou tak, aby nedošlo k jeho vyjetí z dráhy.

Pomocí fyzikálních principů není problém zjišťovat pozici auta na dráze pomocí integrace naměřených hodnot, praktickým problémem ovšem je, že při integraci dochází také k integraci chyb vzniklých při měření, filtrování a zaokrouhlování, dále pak k integraci šumu a offsetu. Po krátké době je pak výsledek prakticky nepoužitelný.

Pro určité zobecnění budeme považovat pohyb auta v ose dráhy jako pohyb v ose x a boční síly působící na auto (normálové zrychlení, odstředivé, nebo dostředivé síly, které nutí auto pohybovat se tímto směrem) jako pohyb v ose y. Pohyb auta po dráze je tedy rovnoměrný nebo rovnoměrně zrychlený a může se pohybovat buď přímočaře (po přímce) nebo po křivce, která je v našem případě tvořena různými (předem definovanými) poloměry zatáček. Pokud se auto pohybuje po přímce, může na něj působit pouze zrychlení ve směru jízdy (v ose x), které neovlivní jeho jízdní vlastnosti. Proto se při přímočarém pohybu může pohybovat libovolně rychle bez jakéhokoliv nebezpečí.

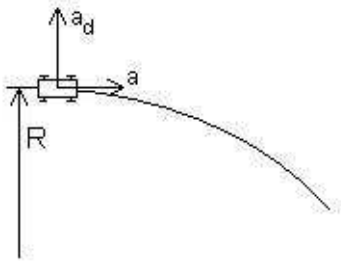
Pro zjednodušení pohybu v zatáčce budeme uvažovat rovnoměrný pohyb, bez zrychlení ve směru jízdy, tedy konstantní rychlostí. V okamžiku vjezdu do zatáčky začne na auto ihned působit normálové zrychlení a_d (Obrázek 2-1). Toto zrychlení je přímo úměrné odstředivé síle F_o :

$$a_d = \frac{V^2}{R} \quad (1)$$

$$F_o = m \cdot \frac{V^2}{R} \Rightarrow F_o = a \cdot m \quad (2)$$

Kde zrychlení a je v [$m.s^{-2}$], rychlost V v [$m.s^{-1}$], F je v [N] a hmotnost m v [kg].

Aby se auto udrželo na dráze, musí být tato odstředivá síla menší než dostředivá síla F_d, která je drží na dráze. Tato dostředivá síla závisí na hmotnosti auta a součiniteli tření kol na dráhu. Tyto hodnoty jsou měřitelné nebo relativně jednoduše zjistitelné, ovšem v našem případě hraje také roli vodící slot, který vede auto v dráze. Jeho vliv není možno jednoduše zjistit, proto je snazší provést měření přímo na dráze tak, abychom zjistili hodnotu normálového zrychlení, kdy se auto ještě udrží na dráze. S těmito pokusy souvisí ještě značné rozdíly mezi jednotlivými drahami, o tom se ale zmíním později.

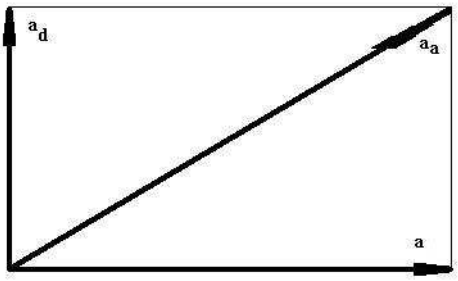


Obrázek 2-1 Tečné a normálové zrychlení působící na auto jedoucí po trajektorii

Tečné a normálové zrychlení je rozkladem celkového zrychlení, působící na auto, viz Obrázek 2-2. Celkové zrychlení pak vyjadřuje vztah (3), vše v [$m.s^{-2}$].

$$a_a = \sqrt{a_d^2 + a^2} \quad (3)$$

Výše uvedené popírá tvrzení účastníků závodu v Rožnově pod Radhoštěm o vhodnosti měnit rychlost v zatáčce, ať se jedná o zrychlování, nebo o brzdění. V obou případech dojde ke zvětšení výsledného zrychlení, změní se pouze znaménko tohoto zrychlení. Pokud pojedeme rovnoměrným pohybem, bude složka a (tečné zrychlení) rovno nule a výsledné zrychlení pak a_d.



Obrázek 2-2 Rozklad zrychlení při pohybu po trajektorii

Dalším stupněm dokonalosti by pak bylo zjišťování projetého úhlu dφ v již zmiňované kartézské soustavě souřadnic, kde d označuje deltu, tj. derivaci podle času. Budeme tedy počítat pohyb po kružnici, kde část kružnice projetá autem je:

$$d\varphi = \frac{d_s}{R} = \frac{V \cdot dt \cdot a_d}{V^2} = \frac{dt \cdot a_d}{V} \qquad (4)$$

Kde dφ je derivace (část) úhlu, ds je derivace dráhy v [m], R je poloměr taktéž v [m], dt je delta času v [s], V rychlost v [$m.s^{-1}$] a ad je normálové zrychlení [$m.s^{-2}$] (podotýkám, že jsme stále u rovnoměrného pohybu).

Pokud výsledný úhel dφ bude 360 stupňů a my budeme opět v bodě B_0, pak jsme projeli celé kolo a začínáme opět od začátku. K tomu ovšem potřebujeme zjišťovat pozici auta v našem souřadném systému. Toho bychom mohli docílit například nějak takto, kde dx nebo dy je delta (posuv) souřadnic v [m], ds je delta (část) dráhy v [m], dt je delta času v [s] sin nebo cos φ je úhel:

$$\begin{aligned} dx &= ds \cdot \cos\varphi = V \cdot dt \cdot \cos\varphi \\ dy &= ds \cdot \sin\varphi = V \cdot dt \cdot \sin\varphi \end{aligned} \qquad (5)$$

Výsledkem (4),(5), pak je vždy delta (určitá část) dráhy, úhlu, nebo jiné veličiny (která nás zajímá) za jednotku času definované právě pomocí dt. Výslednou hodnotu bychom pak obdrželi integrací jednotlivých hodnot. Protože se však pohybujeme v digitální oblasti, která jak všichni víme, není spojitá, můžeme místo integrálů použít sumu, čímž se stejně dopracujeme k žádaným výsledkům. Pokud tedy budeme například sledovat úhel φ a zvolíme-li dt například 10 mS, pak musíme každých 10 mS vzorkovat normálové zrychlení a rychlost. Tyto hodnoty dosadíme do vzorce (4) a budeme je sčítat. Pokud se budeme pohybovat po kružnici, tak sečtením 100 hodnot dostaneme velikost úhlu, který urazilo auto za poslední sekundu. Mimo jiné by tyto hodnoty měly být všechny stejné (stále předpokládáme ustálený rovnoměrný pohyb, po kružnici stálého poloměru a bez chyb měření).

2.2 Možné způsoby měření sil a veličin působících na auto

Na základě předchozí kapitoly jsme si udělali představu, co vlastně potřebujeme měřit. Pokud bych to krátce shrnul, byly by to:

- ✓ Akcelerace **a** (zrychlení) jak v ose x tak v ose y.
- ✓ Čas **t** jízdy auta.
- ✓ Dráha **s,** kterou auto ujelo.
- ✓ Rychlost **V,** kterou auto jede.
- ✓ Úhel **φ,** při jízdě po kružnici, který auto ujelo.
- ✓ Pozici **(x, y)** v souřadném systému, kde se auto momentálně nachází.

Pochopitelně není nutno měřit všechny veličiny, ale čím více jich budeme mít k dispozici, tím přesnější výsledky dostaneme. Na této soutěži máme zjednodušenou práci v tom, že je předem známo z jakých dílů se skládá dráha (přestože jejich přesný počet a uspořádání může být libovolný). Jsou to:

- ✓ Rovinky tří různých délek.
- ✓ Zatáčka 1/60 stupňů.
- ✓ Zatáčka 1/30 stupňů.
- ✓ Zatáčka 2/30 stupňů.
- ✓ Křížení.

Délky rovinek jsou ve specifikaci autodráhy Carrera [8] a také v příloze mé práce, stejně jako velikosti (poloměr) zatáček. Typické provedení zatáčky 2/30 je viz Obrázek 2-3.

Obrázek 2-3 Zatáčka 2/30 Carrera Evolution

Zastavil bych se u číslování zatáček- první číslo určuje velikost poloměru, kde 1 je nejmenší 2 je větší až 4, který je největší. Přitom velikost jednotlivých dílů je volena tak, aby se daly jednotlivé zatáčky spojovat paralelně, tj. spojení zatáčky 1 pro dvě vnitřní dráhy a zatáčky 2 pro dvě vnější dráhy nám vytvoří paralelní čtyřproudou dráhu (viz [9]). Číslo za lomítkem uvádí úhel ve stupních, takže z nám známých hodnot pak jsme schopni spočítat délku oblouku tvořenou kterýmkoliv dílem zatáčky takto:

$$s = \frac{2\pi r}{360^{o}}\alpha \qquad (6)$$

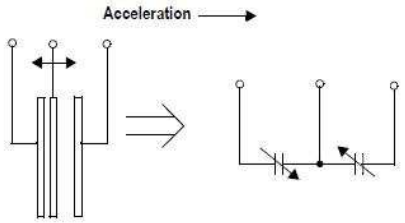
Kde s je dráha v [m], π je konstanta- Ludolfovo číslo, r je poloměr zatáčky v [m] a α úhel ve stupních.

Jak jsem již prezentoval v předchozí kapitole, pro většinu údajů budeme potřebovat integrovat (v našem případě sumarizovat) jednotlivé diskrétní vzorky, získané měřením v co nejčastějších časových intervalech. Toto probíhá vždy v přerušení. Čím častěji budeme vykonávat přerušení a čím bude přerušení delší, tím více budeme zatěžovat mikroprocesor. Potřebujeme tedy dosáhnout co nejkratší doby vykonávání přerušení, proto v něm nelze provádět příliš složité výpočty.

2.2.1 Princip snímání akcelerometrem

Akcelerometr je součástí dodaného plošného spoje a jedná se o moderní součástku s produkce Freescale Semiconductors [7]. Je to tříosý analogový akcelerometr s rozsahem měření +- 1,5g nebo +- 6g. Indikuje velikost zrychlení ve všech třech osách, měří jak dynamické tak statické (gravitační) zrychlení. Je založen na hermeticky uzavřených g- buňkách vyrobených z polysilikonu. Tvoří kondenzátory

s pohyblivou střední deskou, která se vychyluje ve směru zrychlení vždy jedním směrem. Jak se zvětšuje vzdálenost mezi jednou pevnou a střední pohyblivou deskou, tak se stejným způsobem zmenšuje vzdálenost mezi druhou pevnou a střední pohyblivou deskou. Integrovaná elektronika pak měří změnu kapacity, která je úměrná měřenému zrychlení (viz. Obrázek 2-4).



Obrázek 2-4 Zjednodušený model akcelometru

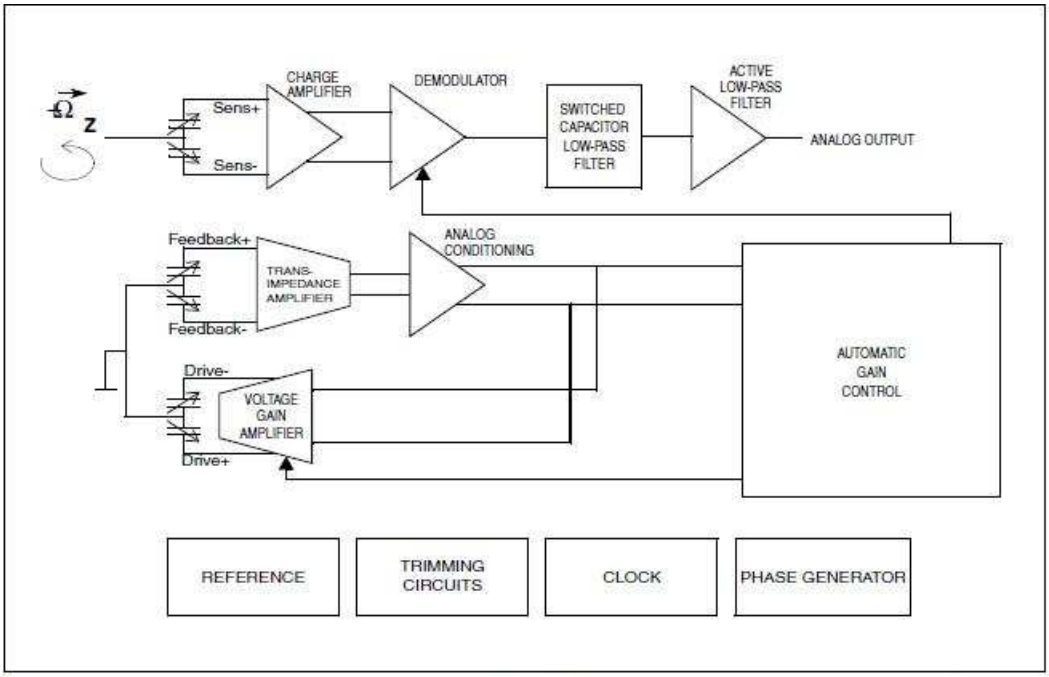
Rozsah měření akcelerometru MMA7361L je určen pinem g- select v rozsahu viz Tabulka 2-1:

g-Select	g-Range	Sensitivity
0	1.5g	800 mV/g
1	6g	206 mV/g

Tabulka 2-1 Rozsah měření akcelerometru MMA7361L

2.2.2 Princip snímání gyroskopem

Gyroskopický senzor LISY300 je jednoosý analogový gyroskopický senzor měřící otáčení v ose Z (yaw) z produkce firmy ST Microelectronics [10]. Výstupem gyroskopického senzoru je úhel otočení ve stupních za sekundu, čímž dostaneme informaci o zatáčení autíčka i v ustálené zatáčce (např. na klopené dráze, nebo v letadle, kde akcelerometr neukáže žádnou výchylku). Gyroskop měří Coriolisovu sílu pomocí vibračního elementu (pizzo krystalu), který funguje jako setrvačník. Ten se v zatáčce snaží setrvávat v rovnoměrném přímočarém pohybu. Coriolisova síla se jej snaží vychýlit, což se snímá a měří podobným způsobem jako zrychlení v akcelerometru (pomocí kondenzátorů se střední pohyblivou deskou) (viz blokové schéma, Obrázek 2-5 a Obrázek 2-4).



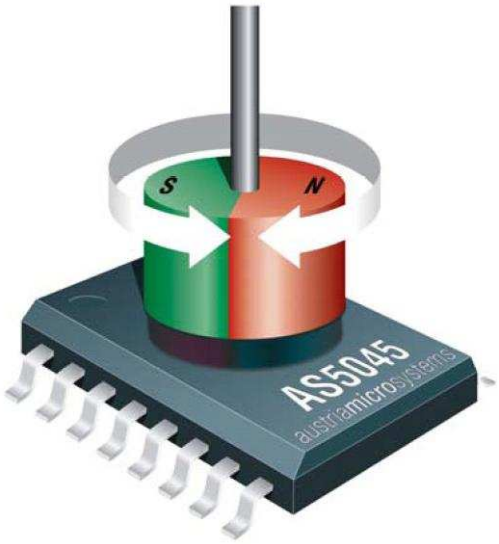
The vibration of the structure is maintained by a drive circuitry in a feedback loop. The sensing signal is filtered and appears as an analog signal at the output.

Obrázek 2-5 Blokové schéma Gyroskopu LISY300AL

Firma ST Microelectronics má s vývojem gyroskopů značné zkušenosti a přestože model LISY300 je jeden z jejich prvních typů, je stále hojně používán. Já jsem pro zjednodušení návrhu a urychlení implementace senzoru do auta použil hotovou destičku Ardupilot Sensor Board Daughter v.1.0, od firmy SparkFun (viz. [6]).

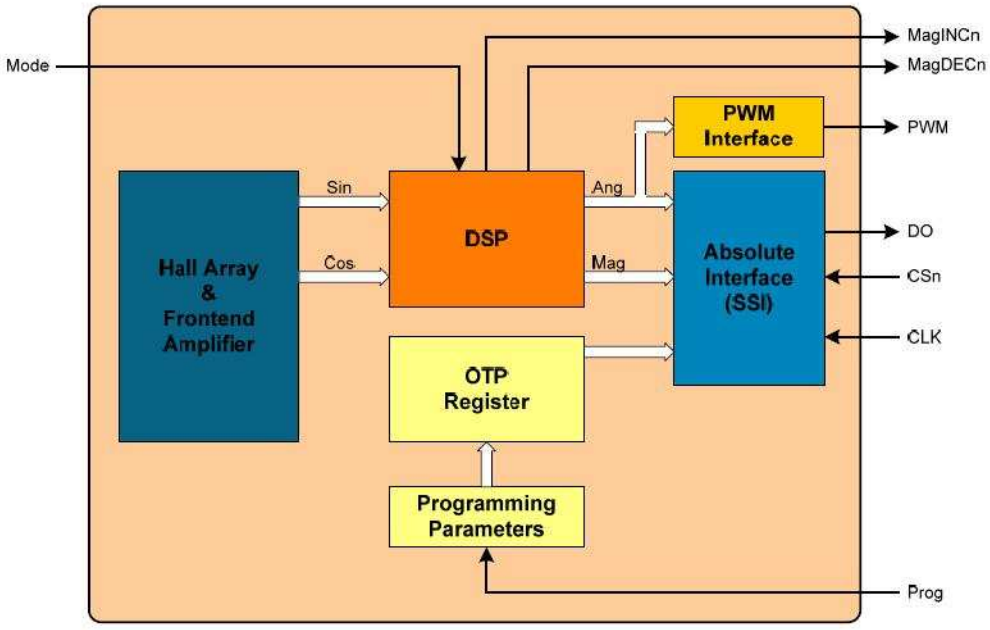
2.2.3 Princip snímání halovým senzorem

Digitální senzor natočení je zajímavý ale málo známý snímač AS5045 z produkce firmy Austria Microsystems [2]. Je založen na poli halových snímačů a v našem autíčku měří úhel natočení slotu. V podstatě poskytuje stejnou informaci jako gyroskopický senzor, s tím rozdílem, že tento snímač poskytuje digitální informaci přímo ve stupních. Pro svoji činnost vyžaduje jednoduchý dvoupólový magnet rotující nad středem čipu (Obrázek 2-6).



Obrázek 2-6 Typické uspořádání AS5045 a magnetu

Senzor je založen na principu snímání magnetického pole pohybující se okolo povrchu čipu. Na čipu je umístěno pole halových snímačů, jejichž výstupem je napěťová reprezentace magnetického pole nacházejícího se okolo povrchu čipu. Toto napětí je převedeno pomocí sigma- delta analogově digitálního převodníku na digitální signál a zpracováno pomocí DSP algoritmu v CORDIC (Coordinate Rotation Digital Computer). Ten pak vypočítá úhel a úroveň signálu halových senzorů (obrázek 2.7).



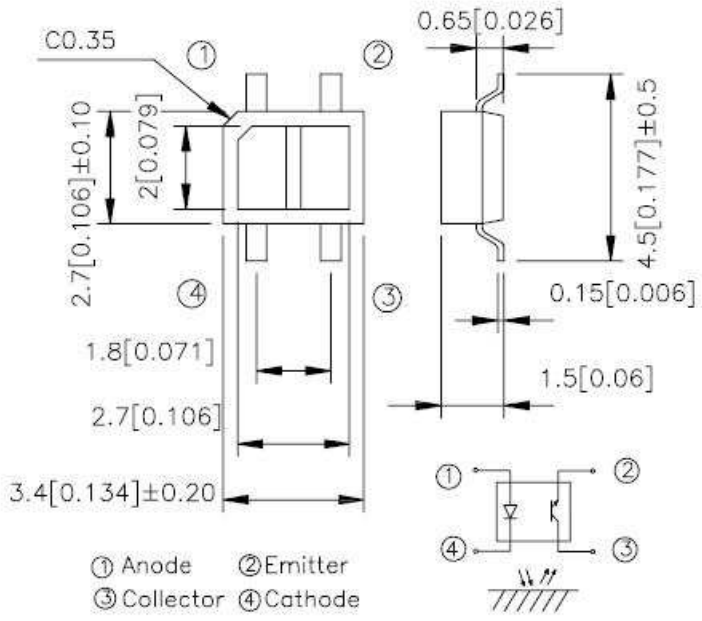
Obrázek 2-7 Blokové schéma rotačního magnetického enkodéru AS5045

Výstupem je 12 bitová absolutní hodnota úhlu otočení ve stupních s krokem 0.0879 stupně, tj. 4096 hodnot na otáčku (360 stupňů), nebo PWM signál. Zajímavé by proto bylo srovnání výstupů z

gyroskopického senzoru a tohoto snímače. Problémem je ovšem nemožnost implementace senzoru tak aby byly splněny podmínky soutěže.

2.2.4 Některé z ostatních možných senzorů

Jako další senzory jsem použil optické snímače, sloužící k informaci o ujeté vzdálenosti a rychlosti. Jsou od firmy Kingbright, typ KTIR0711S [11]. Tento senzor jsem zvolil pro jeho miniaturní rozměry (Obrázek 2-8). Provedení snímače otáček a od toho se odvíjející měření rychlosti je popsáno v kapitole 1. Je umístěn na zadní nápravě auta. Druhý snímač snímá plochu pod autíčkem a v místě kde je startovní a cílová šachovnice (opět bílé barvy) dává informaci o průjezdu kola. Lze tak jednoznačně detekovat průjezd startovním polem, které je na každém okruhu a je pouze jedno. Součástí startovního pole je totiž napájecí přípojka autodráhy. Problémy při implementaci optických snímačů byly hlavně při detekci průjezdu startovním polem. Pokud totiž autíčko projede rychleji tímto dílem, ne vždy zareaguje na bílý pruh, přesto že je tento pruh dostatečně veliký. Navíc tento pruh je vždy z jedné strany plný, ale z druhé (vnitřní) přerušovaný- ve tvaru šachovnice. Proto by bylo řešením snímat tento pruh z obou stran auta.



Obrázek 2-8 Tvar pouzdra, zapojení vývodů a rozměry KTIR0711S

2.3 Omezení dané použitým mikroprocesorem

Použitý mikroprocesor je sice 32 bitový a poměrně výkonný, ale komplexnost dané úlohy vyžaduje například časté volání přerušení s co nejkratší dobou odezvy. Příkladem je měření údajů senzorů, času a generování PWM, které jsou všechny v přerušení. Navíc systém neobsahuje žádný operační systém, proto je na zodpovědnosti vývojáře vytvořit stavový diagram tak, aby byl mikroprocesor schopen vykonávat všechny úlohy včas. Problémem časové náročnosti některých částí programu se budu zabývat v části 3 – Vývoj software.

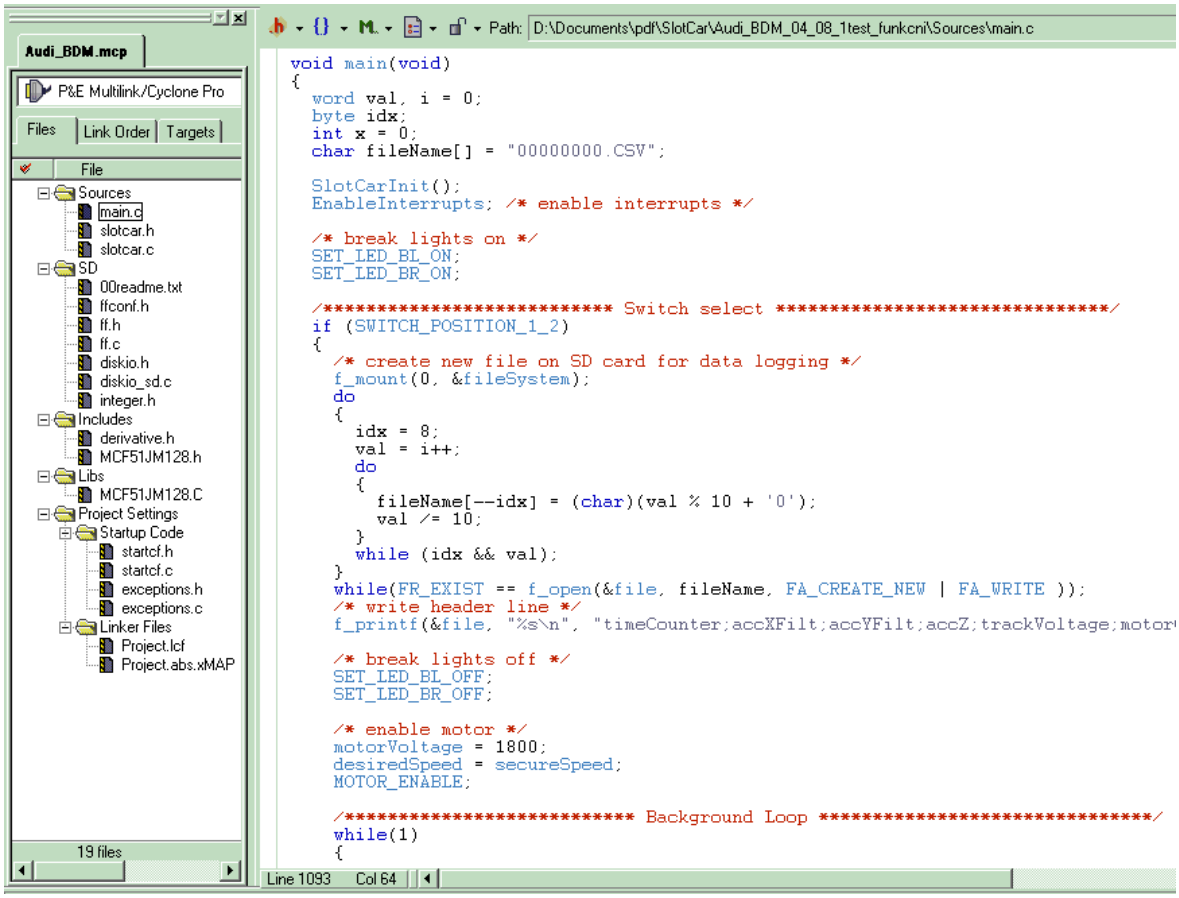
V této aplikaci tedy není základní problém nedostatečný výkon procesoru pro matematické výpočty, ale spíš náročnost neustálé obsluhy přerušení. V tomto kontextu by se dalo říct, že procesor má k dispozici přebytek výkonu pro provádění matematických operací a je možno bez problémů implementovat například další filtry, regulátory nebo další výpočetně složité algoritmy. Takže třeba použití výkonnějších procesorů typu ARM7 (i když dnes již v podstatě zastaralé architektury), nebo nějakého skutečného operačního systému by bylo v této aplikaci kontraproduktivní, protože tyto procesory (nebo operační systémy) mají velmi dlouhou odezvu na přerušení, což by znamenalo, že bychom nebyli schopni dosáhnout tak častých přerušení.

Tento problém mimo jiné právě řeší moderní architektura ARM Cortex M3 (což je přímý konkurent procesorů Coldfire, jmenovitě typů V1 a V2). Z uvedeného vyplývá, že větší výkon neznamená vždy krok kupředu. Přesto bych starší typy procesorů ARM (hlavně ARM9 nebo ARM11) neházel do „starého železa“. Jsou stále velmi oblíbené ve spojení s operačními systémy Linux nebo Windows CE a vyrábí je i firma Freescale Semiconductors.

3 VÝVOJ SOFTWARE

3.1 Základní struktura aplikace

Struktura jednotlivých souborů aplikace je vidět na obrázku 3.1. Složka Sources obsahuje soubory main.c, slotcar.h a slotcar.c. V main je většina kódu aplikace, slotcar.c obsahuje inicializaci a nastavení hardware a slotcar.h definuje proměnné, struktury, atd. V složce SD jsou soubory FAT file systému. Ostatní složky a soubory jsou generovány prostředím Code Warrior. Z nich stojí za zmínku hlavně Project.lcf, který obsahuje inicializaci paměti, hlavně stack, heap a podobně. Zde jsem nastavoval velikost stacku a heapu protože proti vzorové aplikaci ukládám do paměti RAM rozsáhlé struktury dat, které slouží k mapování a vytváření profilu trati, dále pak k jízdě podle takto vytvořené mapy.



Obrázek 3-1 Základní struktura aplikace auta

Celá aplikace se skládá z několika částí, přičemž některé z nich jsou již dodané v balíčku podpory od Freescale Semiconductors. Příkladem je FatFs file systém od Elm Chan viz. [17], nebo USB bootloader. Jak jsem již uvedl, bootloader se mi nepodařilo zprovoznit, proto jsem musel celou aplikaci přeportovat. Důvodem bylo umístění bootloaderu ve spodní části paměti flash, blokování vektorů

přerušení (proto musí být tabulka vektorů relokována do paměti RAM) a jeho spouštění před startem aplikace. Naštěstí je vzorový příklad jednoduchý a hezky blokově uspořádán, takže s ním lze velmi dobře pracovat. Za tento příklad patří můj velký dík Milanovi Brejlovi z Freescale Semiconductors, který jej vytvořil. Portoval FAT file systém na Coldfire V1 a upravil bootloader původně určený pro JM Badge Board.

3.1.1 Hlavní smyčka programu

V hlavní smyčce programu proběhne nejprve inicializace, vytvoření a otevření souboru pro zápis dat a vlastní start aplikace. Ten začíná funkcí `createLapPattern()`. V této funkci zůstává mikroprocesor po dobu cca dvou kol dráhy, tj. v prvním kole provádí mapování a ukládání dat do lokální struktury dat definované uvnitř funkce jako `struct pattern match[256]`. Jednotlivý prvek struktury je buď rovinka, nebo levá či pravá zatáčka. Struktura je definována následovně:

```
struct pattern
{
  unsigned char flag;
  unsigned char length;
  signed char acc;
}
```

Kde flag určuje typ prvku. Původní myšlenka byla, že při mapování trati zde bude uložena přímo informace o tom, o jaký konkrétní typ prvku se jedná, tedy např. levá zatáčka č. 1, pravá č. 2 a podobně. K tomu však z časových důvodů nedošlo, takže momentálně se rozlišují pouze tři typy prvků.

Length je délka, která určuje délku prvku (v jednotkách WHEEL_DIAMETER / 2, kde WHEEL_DIAMETER = 6597 mikrometrů, tj. cca 6,6 centimetru). Length je tedy celková délka zatáčky nebo rovinky a měla by být při každém průjezdu stejnou zatáčkou totožná.

Acc je průměrné normálové zrychlení v této zatáčce zaokrouhlené tak, aby se vešlo do typu signed char. Základní rozsah měření zrychlení je v rozsahu +- 1,5 g, tedy -128 odpovídá (přibližně) hodnotě -1,5 g a +127 odpovídá (opět přibližně) +1,5 g.

Dráha o délce cca 5 metrů mívá typicky 12 prvků struktury, soutěžní trať o délce okolo 15 metrů má pak mezi 30-40 prvky, struktura o 256 prvcích je tedy značně předimenzovaná. To je z důvodu, že

pokud při porovnávání jednotlivých kol závodu dojde k rozdílům (logika systému detekuje smyk jako zatáčku a vloží tuto domnělou zatáčku navíc, nebo v některém kole nedetekuje některou mírnou zatáčku a považuje ji za rovinku), pak má systém možnost porovnávat další kola a „najít se“ v některém z dalších kol. Pokud by až do zaplnění této struktury nedošlo k nalezení mapy dráhy, tak je nastavena bezpečná rychlost auta tak aby dokončilo závod bez vypadnutí. Pokud dojde k nalezení mapy dráhy, je tato mapa překopírována do globální struktury `struct pattern matchPattern` a k okamžitému návratu do funkce main.

Po návratu z funkce `createLapPattern()` je tedy v globální struktuře `matchPattern` mapa dráhy a v globální proměnné `lapMatch` délka této mapy. V tento okamžik máme tedy mapu dráhy i její velikost k dispozici a víme, že jsme přibližně na jejím počátku. Je tedy nutno se synchronizovat se skutečnou pozicí v mapě. Tuto synchronizaci provádím ještě před tím, než auto přejde do režimu jízdy podle mapy. Většinou dojde k synchronizaci na druhém prvku mapy (s indexem 1, protože první prvek má index 0). V tento okamžik se také zkopíruje nalezená mapa do struktury `testMatch`. Následně tedy začne jízda podle mapy, viz výpis č. 1.

Při jízdě podle mapy se vytváří jednotlivé prvky dráhy stejně jako při hledání mapy dráhy, které se porovnávají s mapou a provádí se synchronizace na konci každé zatáčky. Dále se pak tyto prvky průměrují s příslušnou hodnotou uloženou v `testMatch`. Díky tomu ve struktuře `testMatch` zpřesňujeme mapu s každým projetým kolem dráhy. Pro jízdu podle mapy se tedy využívá struktura `testMatch`. Funkce `setSpeedAndCounter()` nastavuje rychlost auta pro zahájený segment (podle indexu pozice v mapě) a pro rovinky vypočítá předstih (okamžik přechodu na bezpečnou rychlost). Tuto hodnotu vrací jako návratovou. Funkce `getValidFlag()` nám pro změnu vrací prvek struktury pattern podle aktuálního zrychlení. Díky ní tedy dostáváme informace o tom, v jakém segmentu se právě nacházíme a můžeme porovnávat skutečnost s mapou. Pokud nesouhlasí současně projížděný segment s tím, který má jet, nastaví se bezpečná rychlost a hledá se kde se auto nachází (pravděpodobně došlo k posunu o jeden segment vpřed nebo vzad).

```
while(1)
{
   test++;
   if(test >= lapMatch)
   {
      test = 0;
      laps++;
   }
   /* and check what type it is. Set speed according this pattern
set testCounter according to measured lap- We need to find when start to brake if
this pattern is long enought, We have to start brake early  */
   testCounter = setSpeedAndCounter(test);
   tempFlag = getValidFlag();
   accTemp = 0;
   accDivider = 0;
   if (tempFlag.flag != matchPattern[test].flag)
   {
/* we are out of pattern, find the right pattern-   previous or next one*/
      desiredSpeed = secureSpeed;
   }
   do
   {
      testFlag = getValidFlag();
      if((testFlag.acc < -10)||(testFlag.acc > 10)) desiredSpeed =
secureSpeed;
   } while ((testCounter > counter) && (testFlag.flag ==
matchPattern[test].flag));
   testMatch[test].acc = tempFlag.acc;
   if(counter >= testCounter)
   {
      desiredSpeed = secureSpeed;
      do
      {
         testFlag = getValidFlag();
      } while(testFlag.flag == matchPattern[test].flag);
   }
   else
   {
      counter2 = counter + 2;
      do
      {
         testFlag = getValidFlag();
      } while ((counter < counter2)&&(testFlag.flag !=
matchPattern[test].flag));
      while(testFlag.flag == matchPattern[test].flag)
      {
         testFlag = getValidFlag();
      }
   }
   patternLength = counter - patternLength;
   testMatch[test].length = (unsigned char)((testMatch[test].length +
patternLength)/2);
   patternLength = counter;
}
```

Výpis č. 1 Jízda podle mapy

3.1.2 Přerušovací subsystém

Jak jsem již uvedl, přerušení a délka jednotlivých rutin je zásadní pro správnou funkci celé aplikace. Využívají se následující rutiny:

- ✓ `interrupt VectorNumber_Vrtc void Vrtc_isr(void)` Rutina přerušení od RTC v intervalu 10 mS. V této rutině se provádí ukládání nalogovaných dat z bufferu paměti do micro SD karty funkcí `disk_timerproc()`. Dále se aktualizuje signalizace světly auta (brzdění, akcelerace, zatáčení, atd.) a uloží se nově naměřené hodnoty do bufferu RAM.

- ✓ `interrupt VectorNumber_Vtpm2ovf void Vtpm2ovf_isr()` Rutina přerušení od časovače/ PWM modulu 2. Je to nejkratší rutina, která se také vykonává nejčastěji- v 500 mikrosekundových intervalech. Zde se inkrementuje časovač (takže nejmenší měřitelný úsek času je 500 mikrosekund), aktualizuje se hodnota PWM motoru a spouští se převod analogového měření.

- ✓ `interrupt VectorNumber_Vadc void Vadc_isr(void)` Rutina přerušení od AD převodníku. Po ukončení analogového převodu je volána tato rutina. První změřená analogová hodnota (jejíž převod se spustil v přerušení řízení PWM) se uloží do globální proměnné a spustí převod další hodnoty. Měří se celkem 6 hodnot- proud motoru, úhel otočení gyroskopem, zrychlení v osách x,y,z a napájecí napětí.

- ✓ `Interupt VectorNumber_Vkeyboard void Vkeyboard_isr()` Rutina přerušení od KBI modulu (nazývá se keyboard interrupt module). Toto přerušení vyvolávají optické snímače měřící otáčky zadních kol a snímající startovní/ cílovou rovinku. Přerušení vyvolané snímačem sledující otáčení zadního kola se provádí průměrně každých 50- 60 ms a je závislé na rychlosti auta. Zde se provádí výpočet rychlosti auta, průměrování normálového zrychlení, aktualizace čítače otáček a vlastní regulace motoru auta.

3.2 Měření, filtrace a ukládání dat

Měření analogových i digitálních vzorků probíhá v přerušení (viz. Přerušovací subsystém).

3.2.1 Způsob měření jednotlivých vzorků

```
/* ADC Conversion Complete Interrupt (sequence of 5 interrupts every
1/2ms)*/
interrupt VectorNumber_Vadc void Vadc_isr(void)
{
  unsigned int output;
  /*
   * Read the new sample using READ_ADC_SAMPLE macro.
   * The reading also clears the interrupt flag.
   */
  switch(ADCSC1_ADCH)
  {
  case MOTOR_CURRENT:
    motorCurrent = READ_ADC_SAMPLE;
    START_CONV(GYRO);
    break;
  case GYRO:
    gyro = READ_ADC_SAMPLE;
    START_CONV(ACC_X);
    gyroFilt = (unsigned short)(gyroFilt - (gyroFilt >> 4) + gyro);
    break;
  case ACC_X:
    accX = READ_ADC_SAMPLE;
    START_CONV(ACC_Y);
    /* 8-Level Half Band filter */
    if ((output = FilterHalfBand8Lynn(accX)) != 0)
    {
      accXFilt = (unsigned short)output;
      accSum += accXFilt;
      accSumCount++;
    }
    break;
  case ACC_Y:
    accY = READ_ADC_SAMPLE;
    START_CONV(ACC_Z);
    /* EWMA filter */
    accYFilt = (unsigned short)(accYFilt - (accYFilt>>4) + accY);
    break;
  case ACC_Z:
    accZ = READ_ADC_SAMPLE;
    if(GET_INT1)
    {
      INT1_TO_ADC_CONNECT;
      START_CONV(TRACK_VOLTAGE_1);
    }
    else if(GET_INT2)
    {
      INT2_TO_ADC_CONNECT;
      START_CONV(TRACK_VOLTAGE_2);
    }
    break;
  case TRACK_VOLTAGE_1:
  case TRACK_VOLTAGE_2:
    INTx_TO_ADC_DISCONNECT;
    trackVoltage = READ_ADC_SAMPLE;
    break;
  }
}
```

Výpis č. 2- Přerušení AD převodníku

Nejdůležitější a nejzajímavější je měření analogových vzorků ADC převodníkem, viz výpis č. 2. Zde se měří proud motoru, který momentálně nevyužívám, ale je velmi dobře využitelný. Příkladem může být zjištění zvýšeného odběru motoru v důsledku zvýšené zátěže auta, což signalizuje vjezd do zatáčky. Díky vodícímu slotu dojde při vjezdu do zatáčky ke zvýšení tření a tím ke zvýšení odběru proudu motorem.

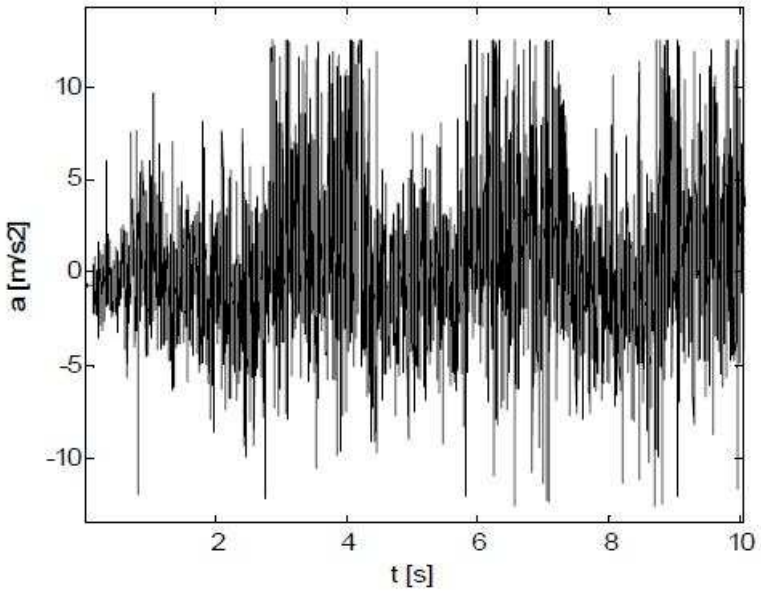
Další měřená veličina je úhel otočení měřený gyroskopem. Použitý gyroskop má maximální měřitelné otáčení 300 stupňů za sekundu, a napětí na výstupu 1,65 V pokud se auto nepohybuje v zatáčce. Citlivost senzoru pak je 3,3 mV na stupeň za sekundu, to znamená, že při otočení auta o 100 stupňů za sekundu bude v průběhu otáčení na výstupu gyroskopu napětí zvýšené (nebo snížené- podle směru otáčení) o 330 mV, proti klidové hodnotě 1,65V, tedy naměříme na něm buď 1,98 V, nebo 1,32 V podle směru otáčení. Z výše uvedeného vyplývá, že gyroskop poskytuje velmi přesnou informaci o úhlu otočení auta a integrací naměřených hodnot získáme informaci o úhlu, které auto projelo.

Následující tři analogově digitální převody je naměřené zrychlení akecelerometrem v osách (x,y,z). V ose x se provádí filtrace osmi úrovňovým Lynnovým half band filtrem, v ose y EWMA filtrem a osa z se nefiltruje, ta se v mé aplikaci nevyužívá. Lynnův a EWMA filtr byly součástí dodané vzorové aplikace. V dalším textu se zastavím u způsobu filtrace, která se mi osvědčila, tj. plovoucí průměr s exponenciálním zapomínáním (EWMA).

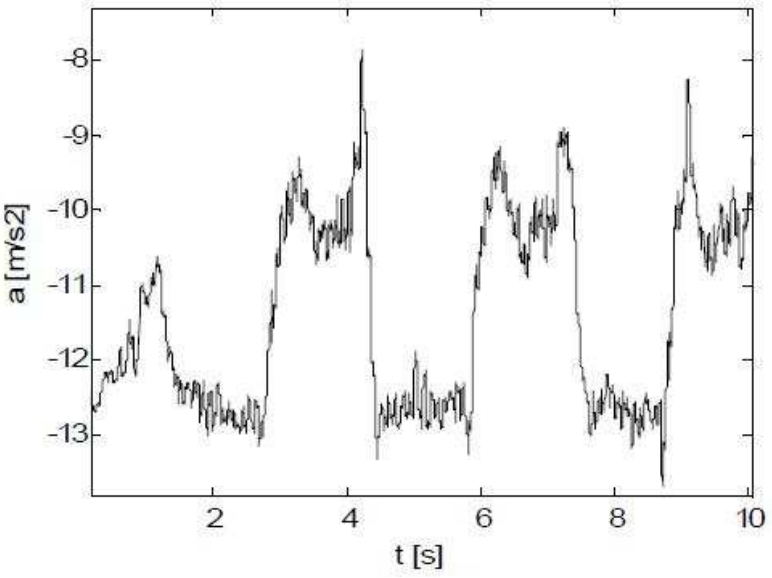
Na měření vstupního napětí je zajímavé hlavně to, že se při každém startu převodu zjišťuje na který vstup (INT1 nebo INT2) je přivedeno napětí, ten se následně přepne do analogového režimu a spustí se převod. Tato komplikace je zde z toho důvodu, že při použití USB bootloaderu slouží tyto vstupy k vyvolání přerušení. Při připojení napájení dojde nejprve k detekci zdroje napájení (USB nebo dráha) a podle toho bootloader zvolí režim mikroprocesoru. Při napájení z USB se mikroprocesor tváří jako USB disk, takže se dá velmi snadno překopírovat hotová aplikace do flash paměti. Při položení auta na dráhu a zapnutí napájení je naopak spuštěna nahraná aplikace.

3.2.2 Použité filtry a jejich implementace, řízení PWM

V předchozí části jsem nastínil použití filtrace při měření a ukládání vzorků. Důležitost filtrace je vidět z následujících obrázků. Obrázek 3-2 ukazuje nefiltrovaný výstup z akcelerometru v ose x na oválné dráze, zatímco Obrázek 3-3 ukazuje signál na totožné dráze, filtrovaný soustavou prvního řádu. Použitý filtr je $G(z) = 0,01 / (z – 0,99)$ s periodou vzorkování $T = 2$ ms ([8]).

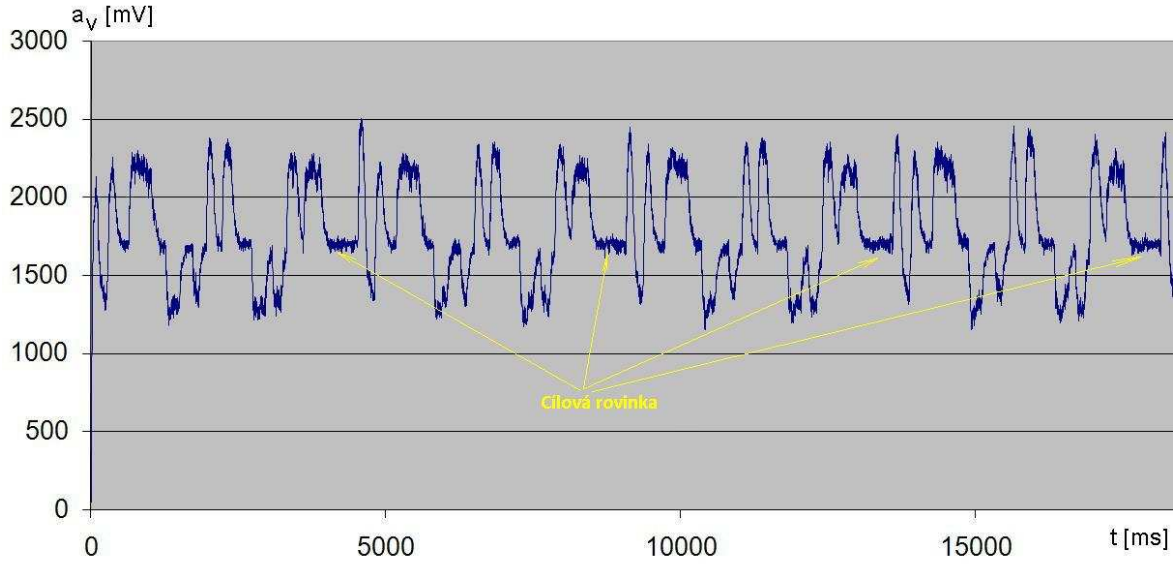


Obrázek 3-2 Nefiltrovaný signál akcelerometru v ose x



Obrázek 3-3 Filtrovaný signál akcelerometru v ose x

Z výše uvedeného je naprosto zřejmá nutnost filtrace dat. Mě osobně se velmi osvědčil plovoucí průměr s exponenciálním zapomínáním. Dále jsem uvažoval o Kalmanově filtru, od kterého jsem byl odrazen poměrně velkou výpočetní náročností tohoto filtru. Nejlepší výsledky podle mého názoru poskytuje právě plovoucí průměr s exponenciálním zapomínáním [6], viz Obrázek 3-4.



Obrázek 3-4 Filtrace plovoucím průměrem s exponenciálním zapomínáním

Obrázek 3-4 ukazuje profil loňské závodní dráhy, filtrovaný plovoucím průměrem s exponenciálním zapomínáním a konstantou k = 10. Ještě bych se zastavil u zvláštní jednotky zrychlení (osy y). Jak jsem již uvedl, výstupem akcelerometru je napětí ve Voltech, kde 1,65V odpovídá nulovému zrychlení, tedy jízdě po rovině. Zrychlení 1g odpovídá 800 mV, které se odečítají nebo přičítají k hodnotě 1,65V (tedy 0g). Maximální rozsah měření pak je 0,45V až 2,85V pro +-1,5g, které měří akcelerometr. Napětí akcelerometru je tedy přímo úměrné velikosti zrychlení auta, mikroprocesor tedy pracuje s hodnotami napětí místo převodu na ms^{-2}.

Plovoucí průměr je průměr z několika posledních měření. Plovoucí pak proto, že delší sekvenci měřených vzorků vytváří okno průměrovaných hodnot, které jakoby „klouže“ za posledním měřením. Můžeme jej vypočítat následovně:

$$avg = \sum_{n=m-k}^{m} y[n] / k \qquad (7)$$

Kde avg je výsledný průměr, y[n] je aktuální měření a k je počet průměrovaných měření. Abychom nemuseli uchovávat k hodnot, je možné počítat průměr rekurzívně avg = (avg*k - y[n-k] + y[n]) / k. Pro plovoucí průměr s exponenciálním zapomínáním (neboli odhad plovoucího průměru) můžeme nahradit odečítání posledních hodnot (tedy jejich „zapomínání“) odečítáním poslední průměrné hodnoty asi takto:

$$avg = (avg*k - avg + y[n]) / k \qquad (8)$$

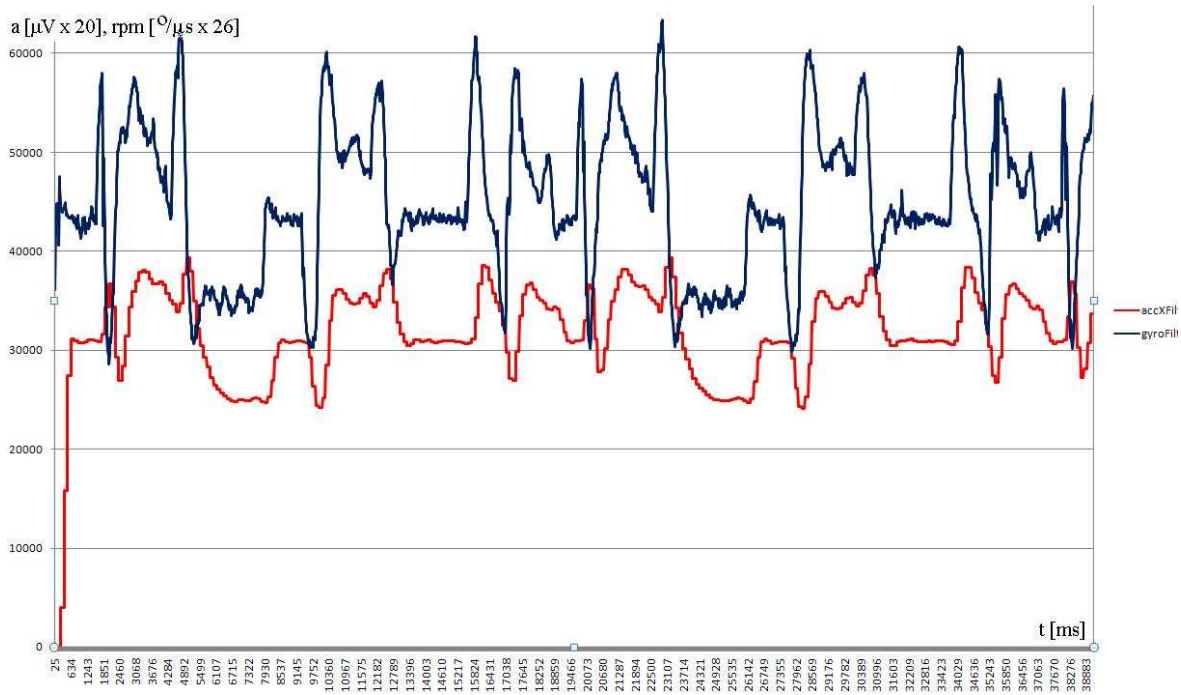
To celé pak můžeme zjednodušit zápisem pomocí korekce:

$$avg = avg + 1/k * (y[n] - avg) \qquad (9)$$

Tím vlastně k předchozímu průměru přičítáme rozdíl mezi předchozím průměrem a naměřenou hodnotou, vynásobenou váhou (převrácená hodnota počtu průměrovaných hodnot- délky okna).

3.2.3 Porovnání vlastností jednotlivých filtrů

S vzorovou aplikací byly dodány výše zmiňované Lynnův a EWMA filtr. Velmi zajímavé výsledky poskytuje Lynnův filtr, který má ale poměrně velké zpoždění (Obrázek 3-5). Díky tomu dostáváme informaci o zrychlení se zpožděním v řádu desítek milisekund, proto je nutno nastavit dostatečný předstih při jízdě podle mapy dráhy. Obrázek 3-5 vidíme profil naší školní testovací dráhy.



Obrázek 3-5 Porovnání signálů gyroskopu a akcelerometru

Červeně jsou zobrazena data z akcelerometru a modře z gyroskopu. Akcelerometr je filtrován pomocí dodaného Half Band Lynnova filtru, gyroskop pomocí EWMA filtru, kde k = 16. Oba senzory měřily trať současně. Kromě jasně viditelného posunu je vidět zaoblení hran křivky akcelerometru, dále pak výraznější a jasnější signál gyroskopu. Jednotky pro zrychlení jsou definovány stejně jako v předchozím případě (Obrázek 3-4),

3.3 Vytváření a hledání profilu trati

Při vytváření profilu trati máme v podstatě dvě možnosti, které využívala většina účastníků závodu (jak jsem se později přesvědčil při diskuzi po ukončení závodu). První možností a asi nejvíce využívanou je prosté zaznamenávání vzorků, například v 10 ms intervalech a pak jejich využívání pro analýzu a jízdu podle nich. Druhou možností je vytváření jednotlivých segmentů dráhy a řízení auta podle mapy dráhy vytvořené tímto způsobem. Tuto možnost jsem využil ve své aplikaci. Je výhodná v tom, že není tak náročná na paměť jako první varianta (jak jsem se již zmínil, typická dráha má mezi 30-40 vzory, to znamená, že ukládáme pouze několik desítek hodnot), ale je nutno ji počítat průběžně. S tím pak také souvisí metoda nalezení mapy dráhy. Pro první variantu je výhodné použít korelační analýzu, i když tuto možnost většina účastníků nevyužila. Druhou metodu nelze tímto způsobem analyzovat, proto je nutno najít jinou vhodnou metodu.

Výpis č. 3 ukazuje způsob nalezení mapy v mé aplikaci. Vstupní podmínka index > 9 zajišťuje minimální délku dráhy 10 segmentů (jinak by docházelo k falešnému nalezení okruhu před skutečným koncem kola). V cyklu do – while se prochází dosud načtená a uložená dráha pomocí vyhledávání hrubou silou. V proměnné index je poslední dosud načtený segment, v search naopak poslední nalezený prvek. Protože prohledávání začne vždy po načtení desátého segmentu (ale dráha bude vždy větší než 10 segmentů), je zde pomocná proměnná restart. díky ní se proměnné inicializují, pouze pokud není nalezeno celé kolo.

Pro správné nalezení trati je tedy nutno nejprve jet konstantní rychlostí, ukládat a vyhodnocovat naměřené vzorky. Po nalezení mapy některým z dále uvedených algoritmů (nebo některým jiným, v mé práci je pochopitelně pouze malý výběr z mnoha možností) pak auto může jet podle této mapy mnohem rychleji. Proto je důležité ji najít co nejdříve.

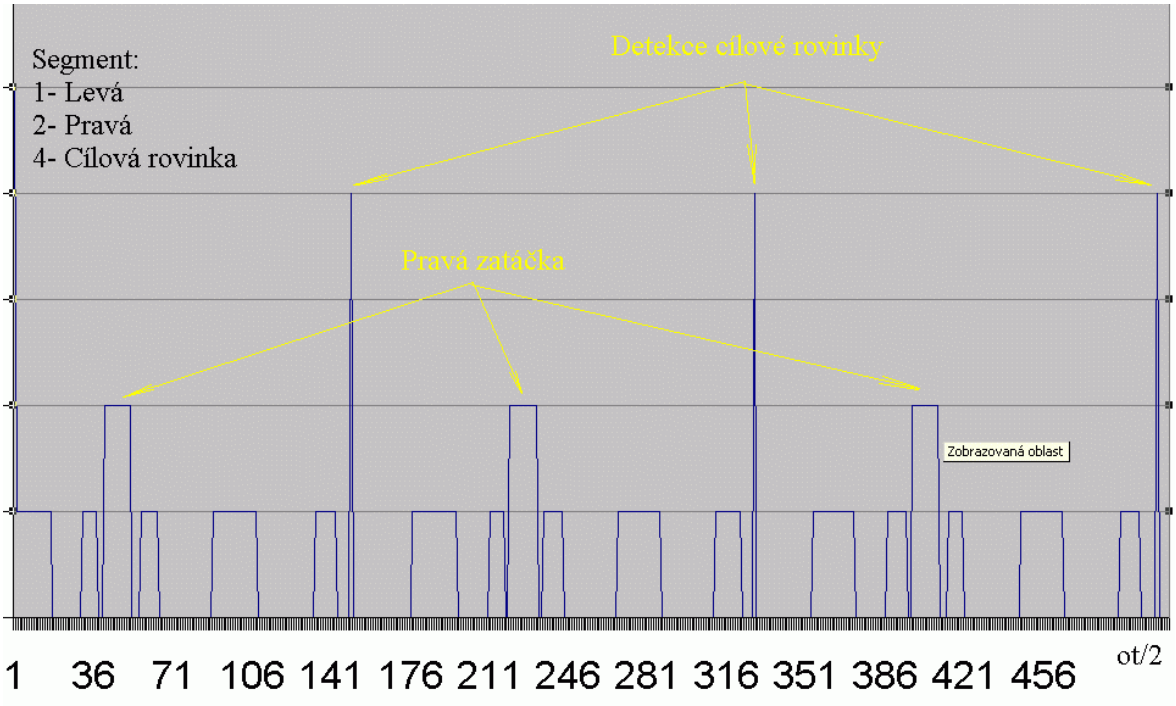
Z tohoto důvodu bylo pro zatraktivnění závodu v letošním roce přidáno pravidlo, že závod začíná za cílovou rovinkou, tedy měření začne až po průjezdu téměř jednoho celého kola. Při jízdě podle mapy zrychlí auto na rovinkách na maximální rychlost (které ovšem téměř nikdy nedosáhne) a před zatáčkou začne brzdit. Zde se tedy projeví (ne)přesnost mapy. Je nutno začít brzdit s dostatečným předstihem aby průjezd zatáčkou byl bezpečný, ale čím déle pojedeme maximální rychlostí, tím vyšší je pak průměrná rychlost na kolo. Ověřil jsem si, že pokud je algoritmus příliš pesimistický („opatrný“) je výsledek kontraproduktivní, takže čas na kolo je delší než jízda konstantní rychlostí.

```
if(index > 9)
{
  /* for finding a lap */
  unsigned static char start = 0;
  unsigned static char search = 9;
  unsigned static char searchBase = 9;
  if(restart == 1)
  {
    start = 0;
    search = 9;
    searchBase = 9;
    restart = 0;
  }
  do
  {
    /* find first lap*/
    if(match[start].flag == match[search].flag)
    {
      start++;
      search++;
    }
    else
    {
      searchBase++;
      search = searchBase;
      start = tempStart;
      if((index > 48) && (searchBase >= index))
      {
        tempStart++;
        if(tempStart > (index - 10))
        {
          restart = 1;
          tempStart = 0;
        }
        else
        {
          searchBase = (unsigned char)(tempStart + 9);
        }
      }
    }
    if (start >= searchBase)
    {
      unsigned char copy = 0;
      for(;start < search; start++)
      {
        matchPattern[copy++] = match[start];
      }
      lapMatch = copy;
      goto lapFound;
    }
  } while(index > search);
}
```

Výpis č. 3- Nalezení mapy pomocí Brute force algoritmu

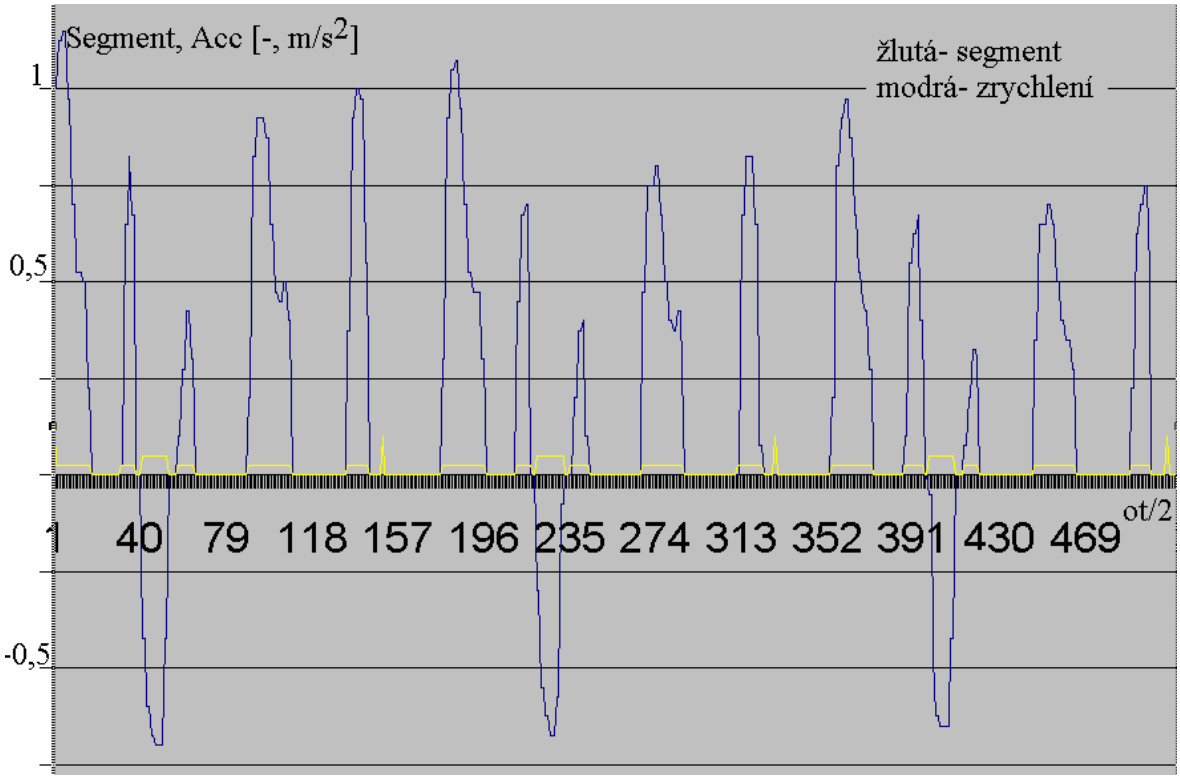
3.3.1 Metoda tvoření celistvých úseků trati

Tuto metodu jsem použil ve své aplikaci. Jakmile je zahájeno mapování dráhy (nebo jízda podle mapy dráhy), čeká se na začátek segmentu. V předchozí části jsem nastínil možnosti jejich výběru, nicméně v tento okamžik budeme předpokládat existenci pouze tří- levá nebo pravá zatáčka a rovinka. Vyhledávání pak probíhá podle normálového zrychlení se zavedenou hysterezí, aby nedocházelo k nalezení segmentu tam, kde ve skutečnosti není. Po nalezení začátku se tedy začnou průměrovat hodnoty, dokud nedojde k nalezení konce. Pak se vytvoří prvek struktury (viz Hlavní smyčka programu), kde je uložen typ, průměrné zrychlení a délka tohoto segmentu. Současně se začne vytvářet další. Výsledek pak vypadá jako na Obrázek 3-6 Výsledná segmentace dráhy. Zde je zobrazen pouze detail segmentů, kompletní data jsou pak zobrazena na Obrázek 3-7 Segmentace dráhy a normálové zrychlení.



Obrázek 3-6 Výsledná segmentace dráhy

Za povšimnutí stojí téměř totožné údaje pro každé kolo, dále pak jednoznačná identifikace cílové rovinky optickým senzorem. Bohužel v praxi se ukázalo, že se na tento údaj nelze spolehnout. Při vyšší rychlosti (při jízdě podle mapy) nebyla tato rovinka detekována se 100% spolehlivostí. Pravděpodobnou příčinou bude nevhodný návrh zesilovače optického senzoru. Pro příští rok se jej pokusím vhodně přepracovat.



Obrázek 3-7 Segmentace dráhy a normálové zrychlení

3.3.2 Prosté měření jednotlivých vzorků při průjezdu tratí

Tato metoda využívá vzorkované měření tak, jak jdou za sebou bez nějakých úprav nebo počítání. Vzhledem k množství dat jsou vždy ukládány na mikro SD kartu. Příkladem takového záznamu může být Obrázek 3-5. K detekci jednotlivých kol dráhy lze pak s úspěchem použít korelační metodu. Princip spočívá v tom, že se porovnává co největší úsek dráhy (počet naměřených vzorků jdoucích za sebou, například 512), s daty uloženými na kartě. Každý nově naměřený vzorek se odečítá od dat uložených na kartě. Jednotlivé výsledky se pak sečtou a za shodu se považuje výsledek s minimálním rozdílem. Zde je nutno citlivě stanovit hranici (treshold) kdy lze data ještě považovat za stejná a kdy už ne. Stejný princip se pak použije i při jízdě podle mapy. Nevýhodou této metody je, že vyžaduje rozsáhlé operace s daty (jejich neustálý přesun, porovnávání velkých bloků dat, atd.). Přesto se ukázala jako nesmírně efektivní. Do budoucna se pokusím realizovat tento způsob detekce pro její praktické srovnání.

3.3.3 Vyhledávání hrubou silou

Měření a ukládání dat je jedna část problému, další je pak najít v takových datech konkrétní profil dráhy. Existuje řada možností jak vyhledávat v textu. Slovem „text“ zde musíme chápat posloupnost znaků v informatickém smyslu slova. V tomto významu je „text“ například sekvence čárek a teček morseovy abecedy, nebo posloupnost bitů přijímaná sériovou linkou. Vyhledávat stejné části takového textu pak můžeme i bez znalosti významu takového textu. Nejjednodušší a také nejoblíbenější je

algoritmus vyhledávání hrubou silou (brute force). Na počátku máme dva indexy, t a v. Index t nám označuje celý text (v našem případě neustálou sekvenci dat snímanou jedoucím autem) a index v je vzor který hledáme. Protože však v tomto případě vzor neznáme (oproti běžným situacím, např. hledání ve slovníku), musíme jej nejprve najít. Zvolíme si tedy index v tak, aby byl dostatečně daleko od počátku měření dráhy (indexu t), ale ne zase moc daleko. To by znamenalo, že musíme projíždět například několik kol, než začne vyhledávání.

Máme tedy nějaký index v, ten má hodnotu řekněme v_0 a index t, který je zatím roven nule. Začneme porovnávat jednotlivé „znaky textu“. Pokud jsou stejné, inkrementujeme oba indexy o jedničku. Tak pokračujeme do nalezení shody, nebo dokud se nezačnou lišit. V případě, že se liší, vracíme se zpět, ovšem o jakou hodnotu? Index v vrátíme na začátek, tedy na v_0 a index t o rozdíl mezi v_0 a v. K výsledku ovšem musíme přičíst jedničku. Takže pokud jde o první pokus, máme nyní index t roven 1. Nyní můžeme začít opět vyhledávat. Kdy ovšem nastane shoda? Za normálních okolností tehdy, když dojdeme na konec vzoru. V našem případě si opět musíme pomoct jinak. Jakmile se začne prohledávaný text znova opakovat, je jasné že projíždíme další kolo. To by mohla být ta správná signalizace konce vzoru. V praxi se ovšem ukázala nutnost použít určité minimální předpokládané délky vzoru, nebo počítat se shodou více znaků našeho textu. Ideální případ by byl projet celé další kolo (tím máme 100% jistotu, že jsme se „našli“), ale tím ztratíme další kolo, protože auto stále jede konstantní rychlostí.

Výsledná mapa dráhy tak, jak je uložená v paměti mikroprocesoru vypadá jako na Obrázek 3-8. Jedná se o tvar dráhy zobrazený na Obrázek 3-7 a Obrázek 3-6. Toto je část debugovacího výpisu do souboru na flash kartu, proto není popsaný význam jednotlivých sloupců. První sloupec má tedy význam „typ zatáčky“, kde číslo 0 je rovinka, 16 je levá zatáčka a 32 pravá. Tyto konstanty byly voleny záměrně, aby šlo rozeznávat zatáčky bitovým maskováním. Další bity pak měly mít význam velikosti zatáčky (1 = zatáčka č. 1, 2= zatáčka č. 2, 4 = zatáčka č. 3 a 8= zatáčka č. 4). Druhý sloupec určuje délku segmentu v polovinách otáčky kola (viz Obrázek 1-3, na převodu kola jsou dvě přerušovací značky). Prakticky lze z tohoto údaje vypočítat délku každého úseku. Například pro první úsek délky 36: 6597/2 * 36 = 118 746 mikrometrů, tj. 11,9 cm. Poslední sloupec pak znamená průměrné normálové zrychlení za daný segment. Číslo je zaokrouhleno tak, aby se maximální výchylka 1,5g vešla do typu signed char (-128… +127).

Global struct, lapMatch = 12		
0	36	0
16	16	-44
0	7	0
32	5	3
0	9	0
16	28	-10
0	14	0
16	40	-30
0	47	0
16	18	-42
0	24	0
16	35	-42

Obrázek 3-8 Nalezená mapa dráhy

3.3.4 Korelační metoda

Tuto metodu jsem rozebíral již v předchozí části textu. Je velmi efektivní, ale citlivá na správné nastavení prahu (tresholdu). Pomocí této metody však lze najít mapu trati dříve, než metodou brute force. Korelační metodu implementoval kolega Tomáš Solarski, proto jsme mohli porovnávat oba algoritmy. Brute force nalezne mapu se 100% jistotou až po projetí druhého kola, zatímco pomocí korelace lze najít mapu mnohem dříve (typicky po první třetině druhého kola). Pochopitelně se nejedná o mapu nalezenou se 100% jistotou, ale je téměř mizivá pravděpodobnost, že by se tak veliký úsek opakoval na dráze ještě jednou. Jedinou výjimkou je vyřazovací turnaj, kde je dráha přesně symetrická. Ale to pochopitelně nevadí, protože auto neví, že projelo pouze polovinu dráhy, může tedy podle jedné mapy projíždět obě poloviny stejně.

3.3.5 Algoritmus Boyer – Moore

Tento algoritmus je velmi efektivní a jednoduchý na pochopení, proto jsem jej zařadil do své práce, přesto, že jsem jej neaplikoval. Tento algoritmus zahájí hledání od konce vzoru, což proti Brute Force nebo korelaci přináší následující výhody:

- ✓ místo hledání shod, hledá neshody (které se statisticky vyskytují mnohem častěji). Pokud při porovnávání zjistíme, že porovnávaný znak není součástí vzoru, můžeme přeskočit celou délku vzoru!

- ✓ posuny vzoru jsou obvykle mnohem větší než 1.

Potřebuje však určitou předkompilaci, aby bylo možno vytvořit tabulku posunů. Počátek je tedy stejný jako v případě Brute Force, s tím, že neporovnáváme první, ale poslední znak vzoru. Pokud se tento znak v celém vzoru nevyskytuje, přeskočíme v hledaném textu celou délku vzoru. Pokud se znak ve vzoru vyskytuje, posuneme jej tak, aby se oba znaky kryly, a porovnávání pokračuje. Pokud se vzor shoduje s textem, máme hledanou sekvenci, jinak posuneme vzor opět dopředu o jeho délku.

Algoritmus je velmi rychlý, ale je efektivní hlavně pro krátké vzory a dlouhou abecedu, což není náš případ (tady by se uplatnilo zvýšení počtu rozlišovaných segmentů, jak jsem původně plánoval, tedy na levou zatáčku 1, levou zatáčku 2…). Z uvedeného vyplývá, že výhody tohoto algoritmu nejsou v našem případě takové, aby se vyplatilo jej implementovat.

3.4 Jízda podle nalezeného profilu trati

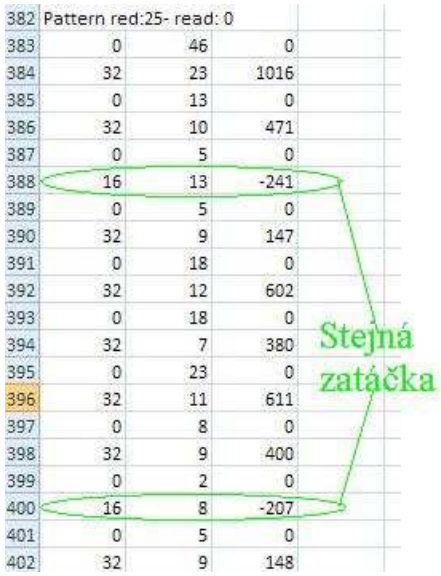
V tento okamžik tedy máme nějakým (pravděpodobně jedním z výše uvedených) způsobem nalezen profil (mapu) trati. Nejprve provedeme synchronizaci, tak jak je popsána výše, následně zahájíme jízdu podle nalezené mapy s tím, že ji průběžně zpřesňujeme průměrováním všech hodnot. Jeden účastník závodu tento systém nazval „jízda slepého tramvajáka“. Takový „tramvaják“ by si také nejprve projel celou trasu, s tím, že by si odměřil rovné úseky a zaznamenal si jak dlouho tyto úseky trvají. Tím by je odlišil od zatáček, které pozná tak že tramvaj „cuká“.

My budeme tedy na rovinkách zrychlovat a předpokládat, že jsme nikde neudělali chybu a nejsme někde jinde. Před zatáčkou pak musíme dostatečně brzy začít brzdit. Čím rychleji jedeme, tím dřív začneme brzdit. Z toho důvodu jsem vytvořil několik stupňů, kdy po překročení určité rychlosti začne auto brzdit s větším předstihem. Synchronizace pak probíhá každou zatáčku, protože víme, jaká zatáčka a jak dlouhá (přibližně) bude. Pokud vyjedeme ze zatáčky, můžeme začít zrychlovat (prakticky můžeme začít již před koncem zatáčky). Problém ovšem nastane, pokud se v zatáčce dostaneme do smyku nebo akcelerujeme příliš rychle. Kola začnou prokluzovat a my dostáváme chybné výsledky (optické snímače detekují určitou projetou dráhu, přestože jsme se dosud nehnuli z místa. Tím dojde k výraznému posunu v pozici na mapě, což je problém. Já jsem tuto situaci řešil tak, že poté, co auto projede rychle rovinku určené délky, zpomalí na bezpečnou rychlost a čeká na zatáčku, kde se opět synchronizuje. Zde by stálo za úvahu detekovat takové úseky a reagovat podle toho, například snížením rychlosti v zatáčce, nebo snížením rychlosti akcelerace.

Dalším problémem, který je nutno vyřešit, je regulace. Většinou se používá některý z P (proporcionální), I (integrační), D (derivační) regulátorů, nebo kombinace výše uvedených. Já jsem použil nejjednodušší verzi proporcionálního regulátoru, což asi také nebyla nejlepší volba.

Pro získávání aktuálních údajů slouží funkce, jejíž prototyp vypadá následovně - `struct pattern getValidFlag();`. Jak je zřejmé, vrací prvek struktury typu pattern. Díky tomu máme možnost jednoduše srovnávat aktuální data s těmi, které jsou uloženy poli mapy dráhy. Na Obrázek 3-9 (kompletní obrázek viz příloha 1) vidíme opět nalezenou mapu dráhy (stejná dráha jako na Obrázek 3-8, tentokrát v opačném směru) a pod ní projeté dvě kola této dráhy. Jedná se o novější verzi (dosud nedokončenou) kde je zrychlení rozšířeno z typu signed char na typ signed short. Předpokládám,

že z důvodu zvýšení přesnosti použiji v budoucnu klasický signed int, nebo změním celou koncepci hledání mapy a jízdy podle ní.



Obrázek 3-9 Příklad mapy a struktury naměřených dat při jízdě (dvě kola) - výřez

4 VYHODNOCENÍ TESTŮ

4.1 Hardware, funkčnost a využitelnost

Celý projekt byl výborně připraven již v počátcích, takže dodaný plošný spoj a výběr komponent hodnotím na výbornou. Drobné připomínky mám k DPS, která je vytvořena v 7 třídě přesnosti, přesto že by ji bylo možno při troše snahy realizovat v 6 třídě. Dále pak byl problém se stabilizátorem – je poměrně málo výkonný, proto jsem jej nahradil výkonnějším a přidal jsem předstabilizaci LM7805 v SMD provedení. Po hardwarové stránce se podařilo celý projekt zvládnout bez problémů. Nepočítám drobné nedostatky, jako je nemožnost startu bootloaderu, které jsem vyřešil použitím BDM interface (což jsem původně stejně plánoval). Gyroskop, stejně jako ostatní senzory se podařilo implementovat bez problémů, i když občas vyžadovaly velmi precizní a hodinářskou práci. Například pro připojení gyroskopu je nutno vyvést analogové vstupy, které na vzorové aplikaci vyvedeny nejsou. Musí se tedy pájet přímo na vývody procesoru v rastru 0.5 mm, což vyžaduje mikroskop. Jediné, na čem musím ještě zapracovat, jsou optické snímače pro detekci cílové rovinky.

4.2 Software, funkčnost a využitelnost

Software je kapitola sama pro sebe. Přesto, že jsem docílil slušných výsledků a mapování dráhy, stejně jako jízda podle této mapy fungovala bez problémů, na závodě v Rožnově pod Radhoštěm auto mapu nenašlo. Útěchou však může být skutečnost, že mi jako jednomu ze tří závodících ani jednou v průběhu závodu nevypadlo auto z dráhy. Zde se projevilo další pravidlo, na které jsme s ostatními soutěžícími přišli- „pokud vše funguje jak má, přeskládej dráhu“. Většinou to znamenalo odhalení nějakého zádrhele.

Dále jsem zjistil, že různé dráhy nejsou stejné. Měl jsem možnost jezdit na třech různých dráhách. Doma, kde jsem nastavil rychlost pro bezpečné projetí zatáčky 120 cm/s (což bylo na hraně), ve škole jsem bez problémů bezpečně projížděl zatáčky rychlostí 145 cm/s. Při závodě v Rožnově pod Radhoštěm pak auto při rychlosti 130 cm/s projíždělo zatáčky bez rizika vypadnutí, ale na konci zatáčky přecházelo do smyku, což ho velmi zpomalovalo a navíc tento smyk bránil nalezení mapy. Při smyku auto detekuje normálové zrychlení jako v zatáčce a protože zde dochází k prokluzu kol, nelze tento smyk detekovat ani délkou projeté dráhy ani časem. Tento smyk je totiž dost dlouhý na to, aby časově odpovídal další zatáčce, a díky prokluzu kol dojde k „protažení dráhy“, což také může odpovídat další zatáčce. Řešením by bylo detekovat otáčky předních kol (nejlépe současně s detekcí zadních) a srovnávat rozdíly. S detekcí otáček předních kol měli zkušenosti některé týmy v Rožnově pod

Radhoštěm, i když zde také docházelo k rozdílům, hlavně při průjezdu zatáčkou různými směry. To pak řešili odpružením předních kol (v normálním autě tento problém řeší diferenciál). Toto odpružení sice přineslo výsledky, ale jednalo se o nepovolenou úpravu, proto ji museli odstranit a uvést auto do původního stavu.

4.3 Plánované změny a úpravy

V současné době plánuji doladění optických snímačů a přidání dalšího snímače na přední kola (jak bylo popsáno v předcházejících částech). Dále předpokládám využití kamery pro detekci dráhy, aby můj „slepý tramvaják“ nebyl zase tak úplně slepý. Pokud bude možno detekovat dráhu pomocí vyhledávání v obraze, nebude třeba vytvářet mapu dráhy a bude možno jet již od počátku prakticky stejně jako podle mapy. Klíčový je výběr vhodného mikroprocesoru, kamery a algoritmu pro detekci vyhledávaného vzoru v obrazu. Mám již nějakou představu o použitelném hardware a částečně jak provádět detekci obrazu, nicméně použitelnost této myšlenky ukáže čas a příslušné pokusy. Předpokládám použití výkonnějších 32 bitových mikroprocesorů (buď z řady Coldfire V2 až V4, nebo některé z ARM Cortex M3). Je totiž nutno zpracovávat velké množství dat v reálném čase, na což bude třeba počítat větším výkonem, než poskytuje doposud použitý Coldfire V1 a s velkou pamětí RAM (předpokládám externí paměť- tedy procesor s EMIF nebo MMU).

ZÁVĚR

Přes nesporné pozitivní výsledky jak v oblasti hardware, tak v oblasti software, není projekt dokonalý jak bych si přál. Svědčí o tom skutečnost, že auto nebylo schopno v průběhu závodu najít mapu dráhy, přestože na jiné dráze se opakovaně nacházelo zcela spolehlivě. Na vině je především problém s optickými snímači, dále pak zaokrouhlování v průběhu výpočtu, atd. Ve své práci uvádím u většiny naznačených problémů také možné způsoby jejich řešení. Tyto nedostatky se pokusím odstranit tak, aby bylo auto na příští závod mnohem lépe připraveno. Dále jej chci vybavit snímáním obrazu dráhy kamerou a detekcí tohoto záznamu. Jeví se mi to jako velká výzva a její úspěšné vyřešení slibuje zajímavé výsledky.

Práce na mém projektu mě velmi nadchla, protože při mém studiu informačních a komunikačních technologií mně chyběla velmi významná část informační oblasti, nazývaná jako embedded systems (nepočítám-li architekturu počítačů v loňském roce). Zabývá se vestavnými systémy od nejjednodušších 8bitových mikrokontrolérů, až po velmi výkonné procesory například ARM Cortex A8, DaVinci od Texas Instruments, nebo DSP procesory, které svými výkony předstihují nejvýkonnější stolní PC. Tyto systémy pracující bez operačních systémů, přes jednoduché operační systémy, realtimové operační systémy až po Linux nebo Windows (většinou v provedení embedded). Zastávám názor, že informatik by měl mít alespoň základní představu o vývoji mikropočítačů v posledních desetiletích, včetně znalosti jejich architektury. Sám jsem začínal na mikroprocesorech řady Intel 8051 programováním v assembleru (správně jazyku symbolických adres) a v mimo školu se pokouším získat znalosti programování v assembleru pro PC. Stále totiž existují algoritmy a procesy které je výhodné nebo přímo nutné psát v tomto jazyce, nebo alespoň v jazyce C se znalostí HW počítače na kterém daný program poběží.

V prvním roce studia jsme měli základy programování javy, což byl můj první vyšší programovací jazyk (pokud nepočítám C pro mikrokontrolery a basic) a byl jsem konsternován přístupem programátorů v tomto jazyce. Při úkolu setřídit dvě pole čísel (každé reprezentované typem double) o délce každého pole v milionech záznamů se nebáli použít rekurzivní algoritmus. No moji námitku, že takhle se každé volání podprogramu ukládá na zásobník, kde je uloženo nejen dané číslo, ale i návratová adresa, registry a další hodnoty, mně bylo řečeno, že se s tím systém nějak popere, a když ne, tak má swapovací soubor. Časem jsem si na tento přístup zvykl, ale vždy si vzpomenu na své začátky v basicu na 8bitových mikropočítačích (Sinclair ZX81 a ZX Spectrum, IQ151, TNS a následně první IBM PC kompatibilní). V tehdejších dobách museli i na PC programátoři přemýšlet co se vlastně v počítači děje a jak ten počítač pracuje.

V průběhu práce na tomto projektu jsem se setkával se stejně zapálenými lidmi (nejen studenty ale i vyučujícími nebo externisty, například z Freescale Semiconductors), kteří tuto soutěž brali jako příležitost zábavnou a soutěživou formou se naučit a dozvědět se něco nového. Proto patří můj velký dík Milanovi Brejlovi z Freescale Semiconductors, který je tažnou silou celé soutěže, a vedoucímu mé bakalářské práce ing. Jiřímu Kotzianovi, Ph. D. za cenné rady a vedení školního kola soutěže FRC. Dále pak všem soutěžícím, se kterými jsem měl možnost se seznámit a vyměnit si zkušenosti a nápady týkající se RFC soutěže.

SEZNAM POUŽITÉ LITERATURY

[1] FREESCALE SEMICONDUCTORS, INC. *MCF51JM Product Summary Page* [online]. c2004 – 2010 [cit. 2010-01-16]. <*http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MCF51JM*>

[2] AUSTRIAMICROSYSTEMS, AG. *AS5045 - 12-bit Magnetic Rotary Encoder* [online]. c2010 [cit. 2010-04-05] Dostupné na WWW: < http://www.austriamicrosystems.com/eng/Products/Magnetic-Encoders/Rotary-Encoders/AS5045>.

[3] BREJL M. – NEČESANÝ J. *Student´s Contest: Self-Driven Slot Car Racing* proceedings of the IMCSIT volume 3, 2008 [cit. 2010-03-20]. ISBN 978-83-60810-14-9 ISSN 1896-7094.

[4] BREJL, Milan. *Freescale Race Challenge 2010* [online]. c2010 [cit. 2010-1-26]. Dostupné na WWW: <http://hw.cz/FRC2010>

[5] SPARKFUN ELECTRONICS. *SparkFun Electronics - ArduIMU Sensor Board - Six Degrees of Freedom (Daughter)* [online]. [cit. 2010-03-28]. Dostupné na WWW: <http://www.sparkfun.com/commerce/product_info.php?products_id=9373>

[6] WINKLER, Zbyněk. *Měření rychlosti* [online]. c2005 [cit. 2010-03-29]. Dostupné na WWW: <http://robotika.cz/guide/filtering/en>

[7] FREESCALE SEMICONDUCTORS, INC. *MMA7361L Product Summary Page* [online]. c2004 – 2010 [cit. 2010-01-23]. <http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MMA7361L&fsrch=1>

[8] CARRERA SLOTS. *Carrera Slots* [online]. c2010 [cit. 2010-04-16]. Dostupné na WWW: <http://carreraslots.com/Merchant5/merchant.mvc?Store_Code=CC&Screen=cp_carrera_help>

[9] SLOT CAR REVIEW. *Carrera Track System* [online]. c2010 [cit. 2010-04-16]. Dostupné na WWW: <http://www.slotcarreview.com/Carrera/Track.html>

[10] ST MICROELECTRONICS. *MEMS inertial sensor: single-axis ±; 300±; /s ana log output yaw rate gyroscope - LISY300AL* [online]. c2010 [cit. 2010-04-05]. Dostupné na WWW: <http://www.st.com/stonline/products/literature/ds/14753/lisy300al.htm>

[11] KINGBRIGHT. *KTIR0711S(Ver.10)* [online]. c2007 [cit. 2010-04-12]. Dostupné na WWW: <http://www.kingbright.com/manager/upload/pdf/KTIR0711S(Ver1189563617.10)>

[12] WRÓBLEWSKI, Piotr. *Algoritmy Datové struktury a programovací techniky.* Computer Press, 2004, 351s. ISBN 80-251-0343-9

[13] Kernighan, Brian, W. – RITCHIE, Denis, M. *Programovací jazyk C.* Tlačiarne SNP, n.p., závod Banská Bystrica, 1989, 256 s. ISBN 80-05-00154-1

[14] HEROUT, Pavel, Doc. Ing. Ph.D. *Učebnice jazyka C.* Kopp, 2007, 271s. ISBN 80-7232-220-6

[15] SCHILDT, Herbert. *Nauč se sám C.* SoftPress, s.r.o., 2001, 620s. ISBN 80-86497-16-X

[16] BARTSCH, Hans-Jochen. *Matematické vzorce.* Academia, 2006, 832s. ISBN 80-200-1448-9.

[17] CHAN, Elm. *ELM - FAT File System Module.* [online]. C2008 [cit. 2010-05-01]. Dostupné na adrese: http://elm-chan.org/fsw/ff/00index_e.html

[18] BORENSTEIN, L., EVERETT, H. R., FENG, L. [i]Where am I? Sensors and Methods for Mobile Robot Positioning.[/i] J. Borenstein. [s.l.] : [s.n.], 1996. 282 s.

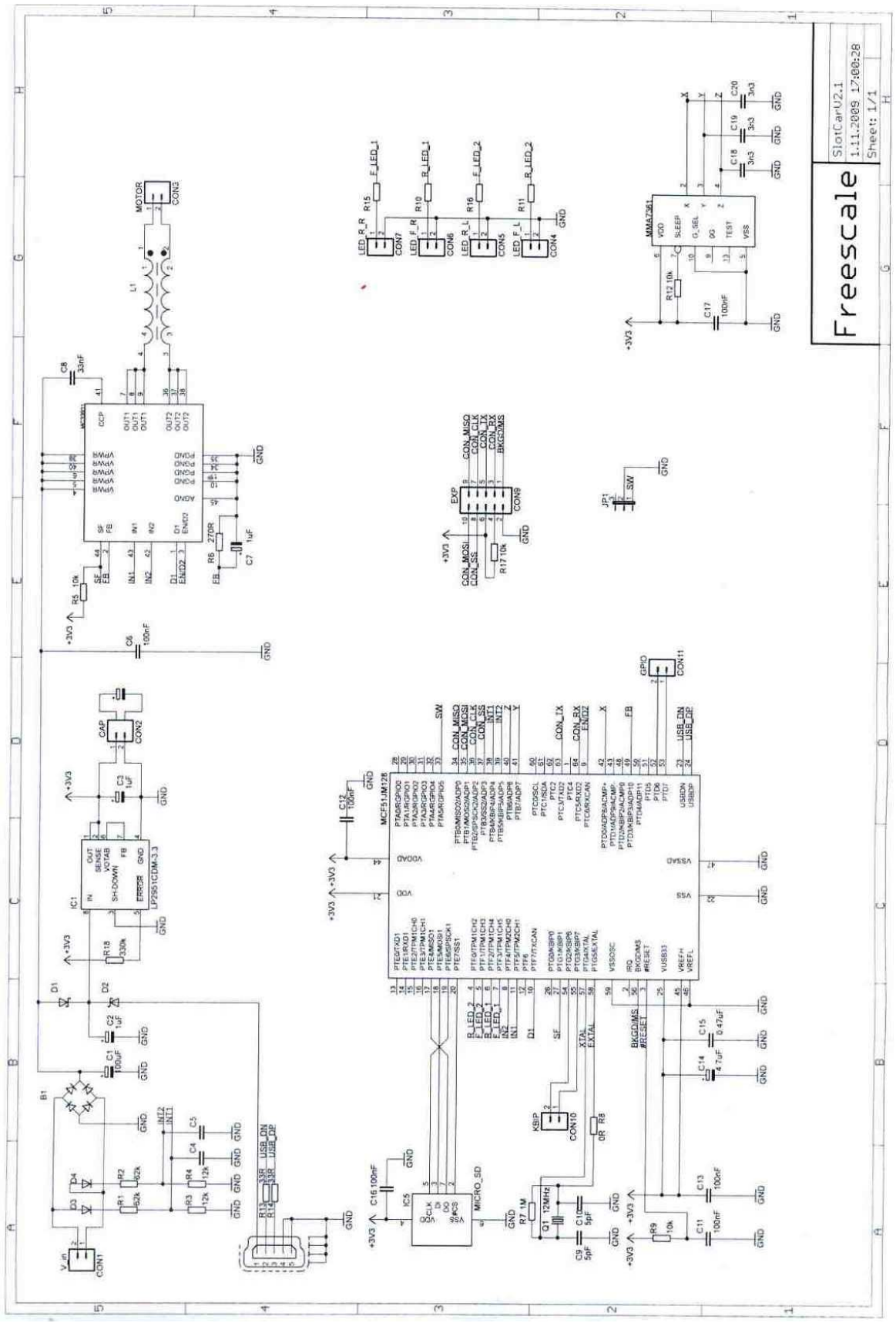
SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

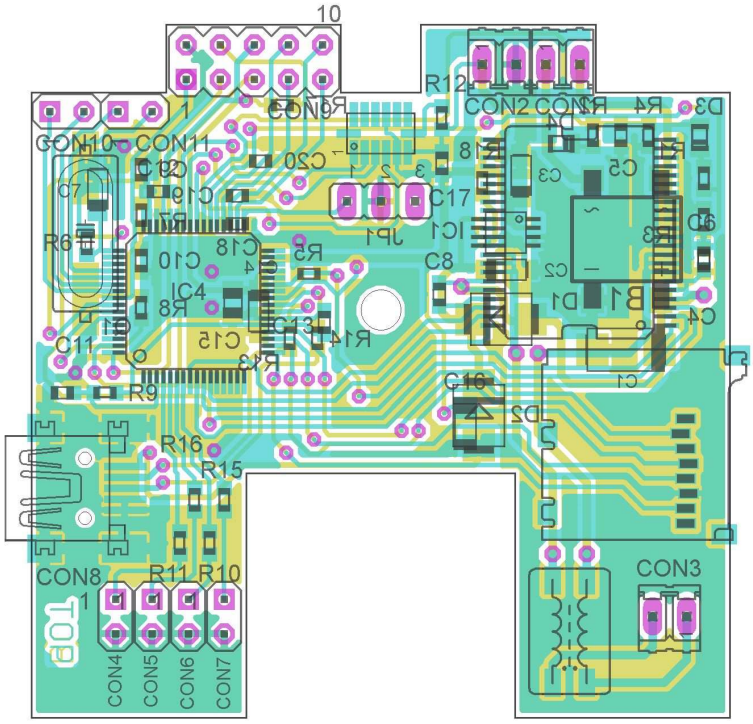
Příloha 1- Příklad mapy a struktury naměřených dat (dvě kola)

363	start : 0, end: 12			
364	0			
365	32			
366	0			
367	32			
368	0			
369	16			
370	0			
371	32			
372	0			
373	32			
374	0			
375	32			
376	0			
377				
378	endLap = 12, indexText = 12, index = 13			
379				
380				
381				
382	Pattern red:25- read: 0			
383	0	46	0	
384	32	23	1016	
385	0	13	0	
386	32	10	471	
387	0	5	0	
388	16	13	-241	
389	0	5	0	
390	32	9	147	
391	0	18	0	
392	32	12	602	
393	0	18	0	
394	32	7	380	
395	0	23	0	
396	32	11	611	
397	0	8	0	
398	32	9	400	
399	0	2	0	
400	16	8	-207	
401	0	5	0	
402	32	9	148	
403	0	23	0	
404	32	23	1011	
405	0	24	0	
406	32	13	675	
407	0	36	0	

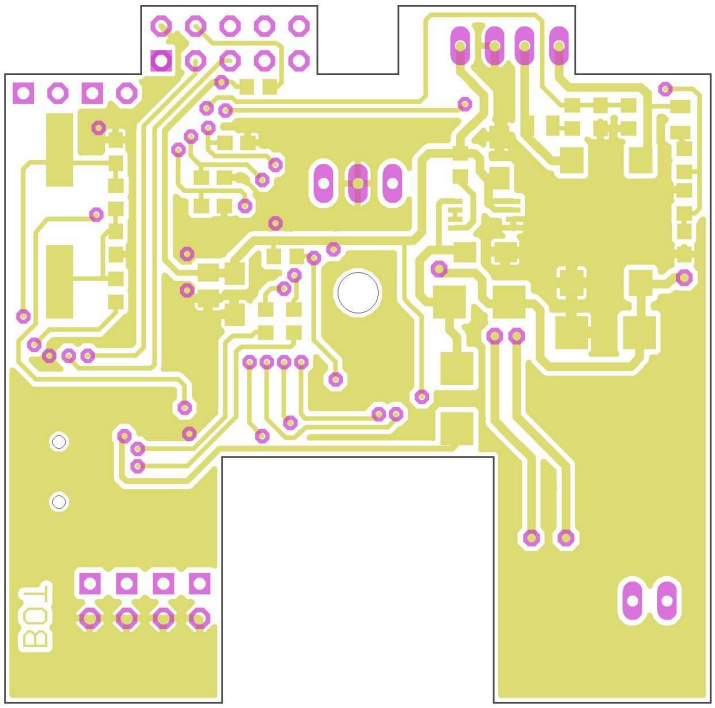
Příloha 2 – Schéma zapojení řídícího mikropočítače auta



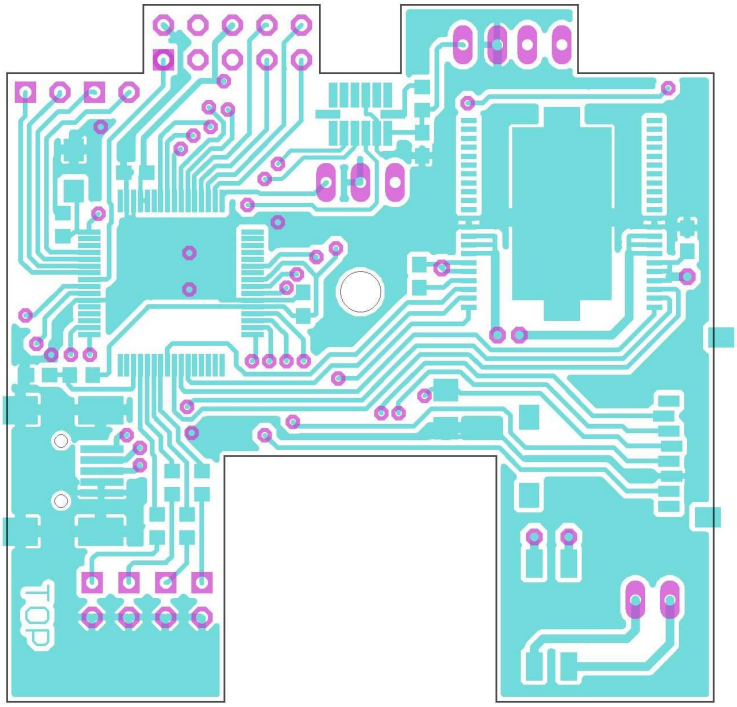
Příloha 3 – DPS řídícího mikropočítače – všechny vrstvy



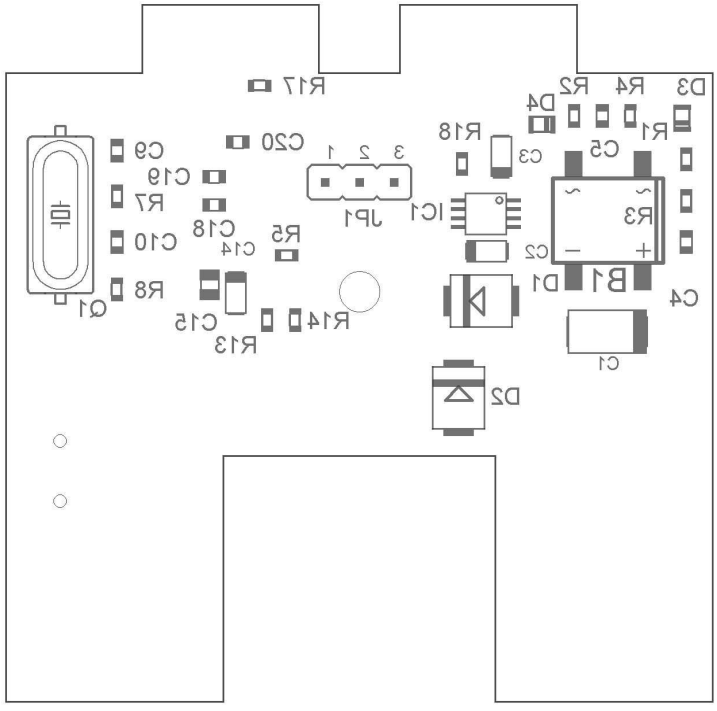
Příloha 4 – Spodní strana DPS řídícího mikropočítače- měď



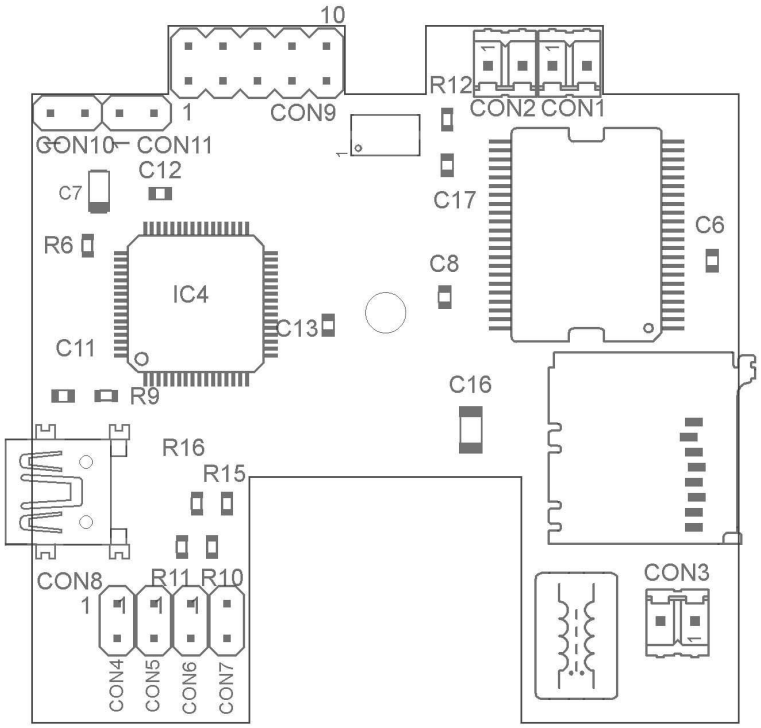
Příloha 5 – Horní strana DPS řídícího mikropočítače- měď



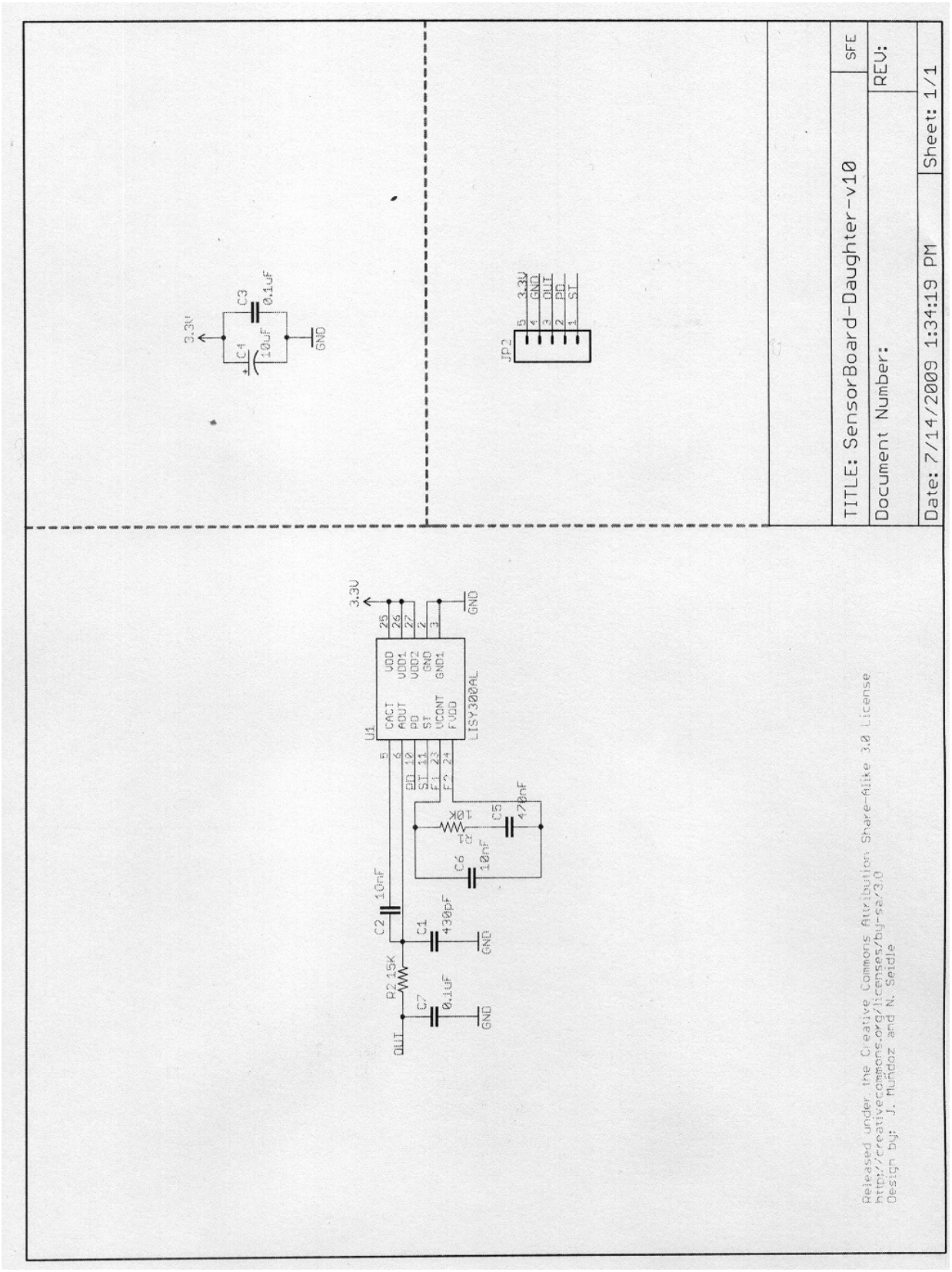
Příloha 6 – Spodní strana DPS řídícího miktopočítače- rozmístění součástek



Příloha 7 – Horní strana DPS řídícího mikropočítače- rozmístění součástek



Příloha 8 – Schéma zapojení modulu gyroskopu



Příloha 9 – Carrera Track Measurements

Item No.	Description	Length (mm)	Length (in)	Radius***
2020509	Standard Straight	345 mm	13.58 in.	---
2020515	Connection Section (incl. 1 standard straight)	345 mm	13.58 in.	---
2020516	Narrow Section (Chicane)	345 mm	13.58 in.	---
2020517	Lane Changing Section	345 mm	13.58 in.	---
2020545	Crossing	---	---	---
2020551	Inside Shoulder, Curve 1/60°	---	---	RO200
2020560	Outside Shoulder, Straight	345 mm	13.58 in.	---
2020561	Outside Shoulder, Curve 1/60°	---	---	RI400/RO500
2020562	Outside Shoulder, Curve 2/30°	---	---	RI600/RO700
2020563	Outside Shoulder, Curve 3/30°	---	---	RI800/RO900
2020564	Outside Shoulder, Bank Curve 1/30°	---	---	RI400/RO500
2020565	Outside Shoulder, Bank Curve 2/30°	---	---	RI600/RO700
2020566	Outside Shoulder, Bank Curve 3/30°	---	---	RI800/RO900
2020567	Outside Shoulder, Curve 1/30°	---	---	RI400/RO500
2020568	Outside Shoulder, Curve 4/15°	---	---	RI1000/RO1100
2020569	Inside Shoulder, Bank Curve 1/30°	---	---	RO200
2020571	Curve 1/60°: 3 pcs. = 180°	---	---	RI200/RO400
2020572	Curve 2/30°: 6 pcs. for 4-lane ext.= 180°	---	---	RI400/RO600
2020573	Curve 3/30°: 6 pcs. for 6-lane ext.= 180°	---	---	RI600/RO800
2020574	Banked Curve 1/30°: 6 pcs. = 180°	---	---	RI200/RO400
2020575	Banked Curve 2/30°: 6 pcs. for 4-lane ext.= 180°	---	---	RI400/RO600
2020576	Banked Curve 3/30°: 6 pcs. for 6-lane ext.= 180°	---	---	RI600/RO800
2020577	Curve 1/30°: 6 pcs. = 180°	---	---	RI200/RO400
2020578	Curve 4/15°: 12 pcs. for 8-lane ext. = 180°	---	---	RI800/RO1000
2020611	1/3 Straight	115 mm	4.53 in.	---
2020612	1/4 Straight	86 mm	3.39 in.	---

Notes: *** RI = Radius of inside edge of track, in mm
RO = Radius of outside edge of track, in mm

1 . Track Width: Nominal: 200 mm / Actual: 198 mm = 7.79 in.
2. Shoulder Width: Nominal: 100m mm / Actual: 99 mm = 3.89 in.
3. Key to Curve Designations (Curve # / # of Degrees):
   Curve 1 makes the smallest diameter of circle
   Curve 2 fits directly around Curve 1
   Curve 3 fits directly around Curve 2
   Curve 4 fits directly around Curve 3
4. # of Degrees indicates what portion of a circle each piece of track makes.

Example: Curve 1/60° is a 60° segment (out of a 360° circle) of Curve 1, the smallest curve.